Date

Expt. No. 01

Page No. 01

Aim:- To study about Software Engineering Principles:

Software Engineering Principles:

- A collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis.

- Populates a software process model with the necessary technical and management how-to's to get the job done.

- A philosophical foundation that guides software practice.

- Technology-specific knowledge has about 3 year half-life
  - Half of what we know today will be obsolete within 3 years.

- In contrast, Software Enginoering principles are likely to serve a professional programmer throughout his or her career.

Teacher's Signature :

Core Principles: Process

- Principle 1: Be agile.
- Principle 2: focus on quality at every step.
- Principle 3: Be ready to adapt.
- Principle 4: Build an effective team.
- Principle 5: Establish mechanisms for communication and coordination.
- Principle 6: Manage change.
- Principle 7: Assess risk.
- Principle 8: Create work products that provide value for others.

Core Principles: Practice

- Principle 1: Divide and conquer.
- Principle 2: Understand the use of abstraction.
- Principle 3: Strive for consistency.
- Principle 4: focus on the transfer of information.
- Principle 5: Build software that exhibits effective modularity
- Principle 6: Look for patterns.
- Principle 7: When possible, represent the problem and its solution from a number of different perspectives.
- Principle 8: Remember that someone will maintain the software.

Communication Principles:

- Poinciple 1: Listen.
- Principle 2: Prepare before you communicate
- Principle 3: Someone should facilitate the activity.
- Principle 4: Face-to-face communication is best.
- Principle 5: Take notes and document decisions.
- Principle 6: Strive for collaboration.
- Principle 7: Stay focused; modularize discussion.
- Principle 8: If something is unclear, draw a picture.
- Principle 9: (a) Once you agree, move on.
  (b) If you cannot agree, move on.
  (c) If a feature or function is unclear and cannot be clarified at the moment, move on.
- Principle 10: Negotiation is not a contest or a game. It works best when both parties win.

Planning Principles:

- Principle 1: Understand the scope of the project.
- Principle 2: Involve stakeholders in the planning activity.
- Principle 3: Recognize that planning is iterative.
- Principle 4: Estimate based on what you know.
- Principle 5: Consider risk as you define the plan.
- Principle 6: Be realistic.
- Principle 7: Adjust granularity as you define the plan.
- Principle 8: Define how you intend to ensure quality.
- Principle 9: Describe how you intend to accomodate change.
- Principle 10: Track the plan frequently and make adjustments as required.

Aim:- To study about Software Development Life Cycle (SDLC)

SDLC (Software Development Life Cycle) is a process used by the software industry to design, develop & test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.

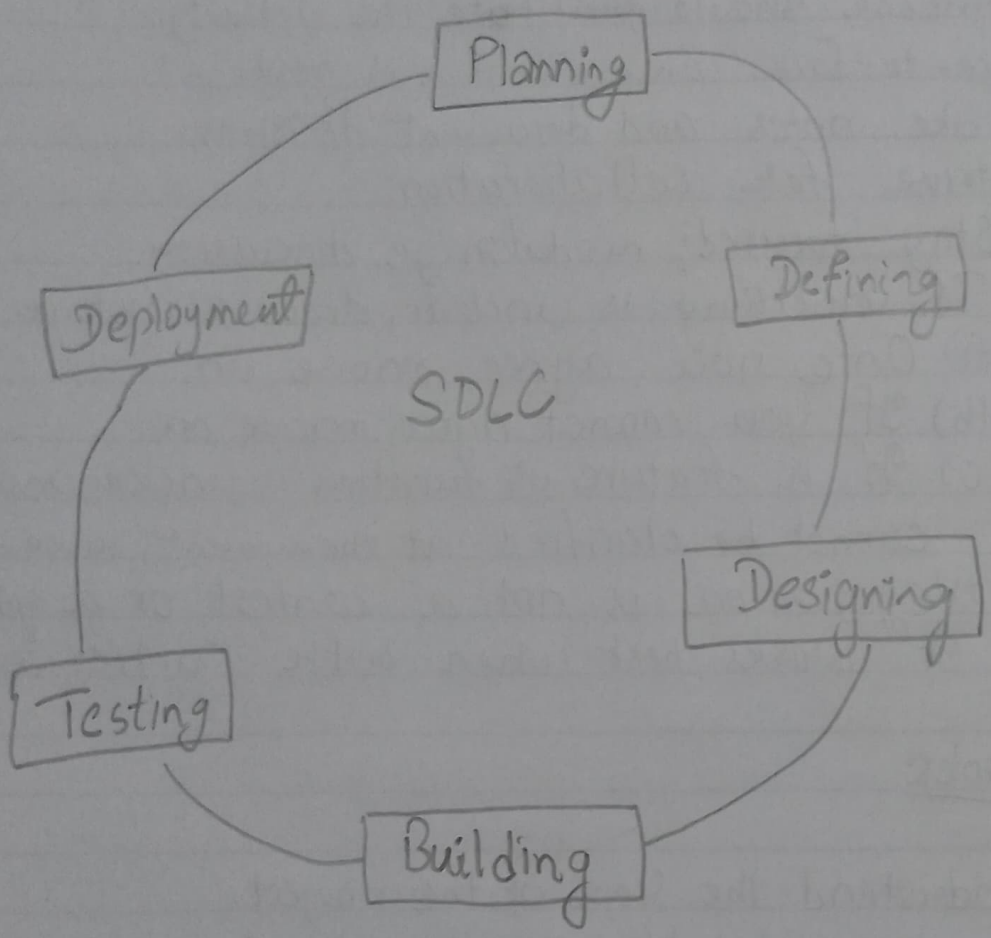
- A typical Software Development Life Cycle consists of the following stages:-

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements

Graphical representation of the various stages of a typical SDLC:-



and identifications of the risk associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with the minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed & developed during the project life-cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness,

design modularity, budget & time constraints, the best design approach is selected for the product.

Stage 4: Building or Developing the Product:

In this stage of SDLC the actual development starts & the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developer must follow the coding guidelines defined by their organisation and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming lang. such as C, C++, Pascal, Java & PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product:

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of the organisation. The product may first be released in a limited segment and tested in the real business environment (UAT - User acceptance Testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

Aim: To study about Software Modelling Principles

- Principle 1: The primary goal of the software team is to build software, not create models.
- Principle 2: Travel light – don't create models more than you needed.
- Principle 3: Strive to produce the simplest model that will describe the problem of the software.
- Principle 4: Build models in a way that makes them amenable to change.
- Principle 5: Be able to state an explicit purpose for each model that is created.
- Principle 6: Adapt the models you develop to the system at hand.
- Principle 7: Try to build useful models, but forget about building perfect models.
- Principle 8: Don't become dogmatic about the syntax of the model. If it communicates content successfully, representation is secondary.
- Principle 9: If your instincts tell you a model isn't right even though it seems OK on paper, your probably have reason to be concerned.

Modelling Principles: Requirements:

- Principle 1: The information domain of a problem must be presented and understood.
- Principle 2: The functions that the software performs must be defined.

- Principle 3: The behaviour of the software must be represented.
- Principle 4: The models that depict information, function, and behaviour must be partitioned in a manner that uncover detail in a layered fashion.
- Principle 5: The analysis task should move from essential information towards implementation detail.

Modelling Principles: Design

- Principle 1: Design should be traceable to the requirements model.
- Principle 2: Always consider the architecture of the system to be built.
- Principle 3: Design of data is as important as design of processing.
- Principle 4: Interfaces must be designated.
- Principle 5: User interface design should be tuned to the needs of the end user. However, in every case, it should stress ease of use.
- Principle 6: Component-level design should be functionally independent.
- Principle 7: Components should be loosely coupled to one another and to the external environment.
- Principle 8: Design representations should be easily understandable.
- Principle 9: The design should be developed iteratively.

With each iteration, the design should strive for greater simplicity.

Coding Principles: Preparation

- Principle 1: Understand the problem you're trying to solve.
- Principle 2: Understand the basic design principles and concept.
- Principle 3: Pick a programming language that meets the need of the software to be built and the environ-ment in which it will operate.
- Principle 4: Select a programming environment that provides tools that will make your work easier.
- Principle 5: Create a set of unit tests that will be applied once the component you code is completed.

Coding Principles: Programming

- Principle 1: Constrain you algorithms by following structured programming practice.
- Principle 2: Consider the use of pair programming.
- Principle 3: Select data structures that will meet the needs of the design.
- Principle 4: Understand the software architecture and create interfaces consistent with it.
- Principle 5: Keep conditional logic as simple as possible.
- Principle 6: Create nested loops in a way that makes them easily testable.

- Principle 7: Follow coding standards
- Principle 8: Write code that is self-documenting.
- Principle 9: Create a visual layout that aids understanding.

Coding Principles: Validation

- Principle 1: Conduct a code walkthrough when appropriate
- Principle 2: Perform unit tests and correct errors you've discovered.
- Principle 3: Refactor the code.

Testing Principles:

- Principle 1: All tests should be traceable to customer requirements.
- Principle 2: Tests should be planned long before testing begins.
- Principle 3: The Pareto principle applies to software testing.
- Principle 4: Testing should begin "in the small" and progress toward testing "in the large".
- Principle 5: Exhaustive testing is not possible.

Deployment Principles:

- Principle 1: Customer expectations for the software must be managed.
- Principle 2: A complete delivery package should be assembled & tested.
- Principle 3: A support regime must be established before the software is delivered.

- Principle 4: Appropriate instructional materials must be provided to the user.
- Principle 5: Buggy software should be fixed first, delivered later.

Aim: To study about Entity-Relationship Diagram (with examples)

Entity-Relationship Diagram:-
ER-modelling is a data modelling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this E-R modelling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

Purpose of ERD:-
- The database analyst gains a better understanding of the data to be contained in the database through the step of construction of the ERD.
- The ERD serves as a documentation tool.
- Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Components of ER-Diagram:

1. Entity: An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in ER diagram.
for e.g. in a school database, students, teachers, classes & courses offered can be treated as entities.

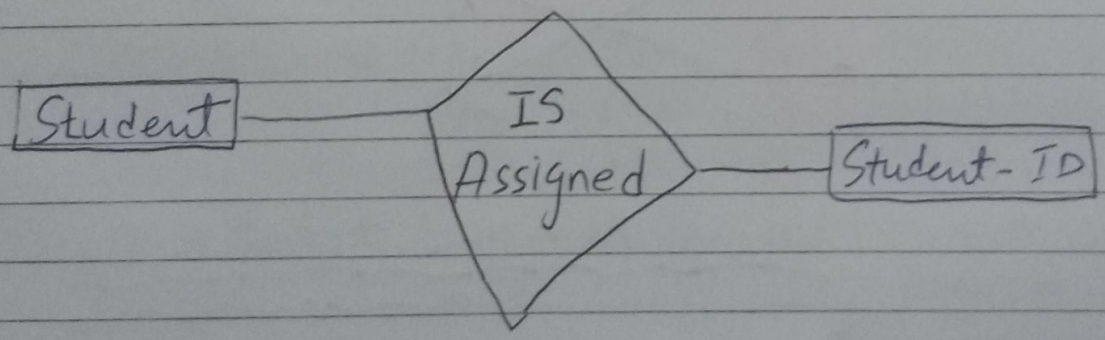
2. Attributes: Entities are denoted utilizing their properties known as attributes. All attributes have values for e.g. a student entity may have name, class, & age as attributes.

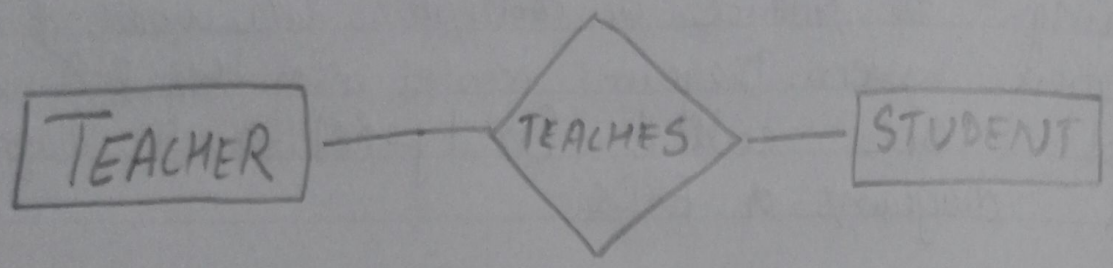
3. Relationship: The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For eg. an employee works at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.

Cardinality: The number of entities present in the entity set is called cardinality of a set.

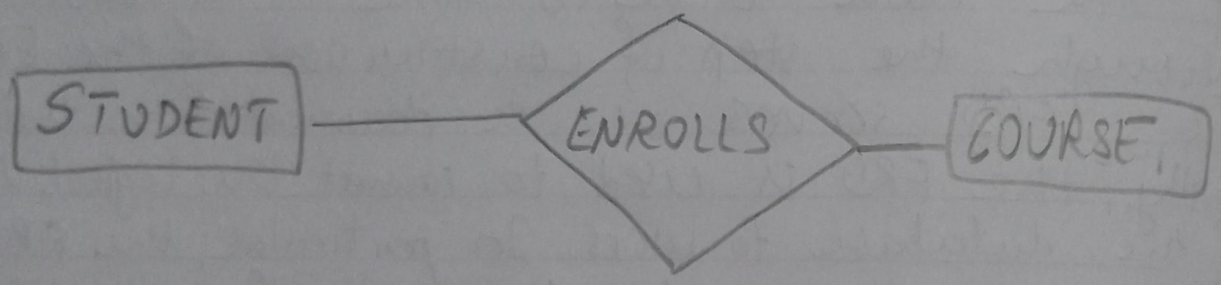
Types of Cardinalities:

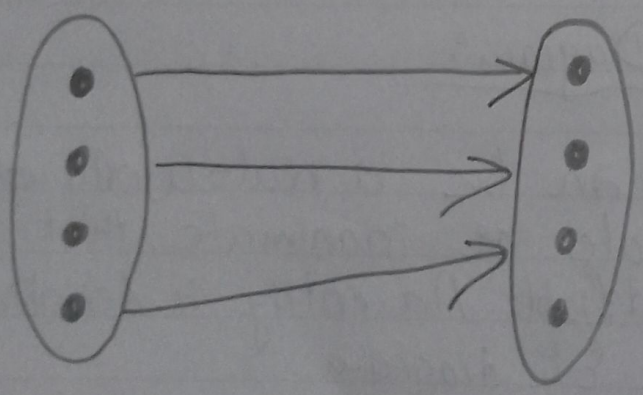
1. One-to-One: One entity from entity set A can be contained with at most one entity of entity set B & vice-versa. let us assume that each student has only one student ID, & each student ID is assigned to only one person so, the relationship will be one-to-one.





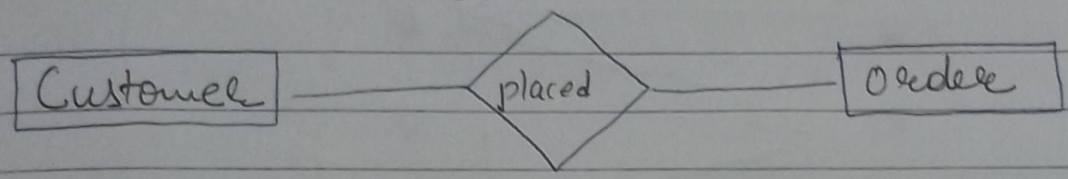
Relationships in ERD



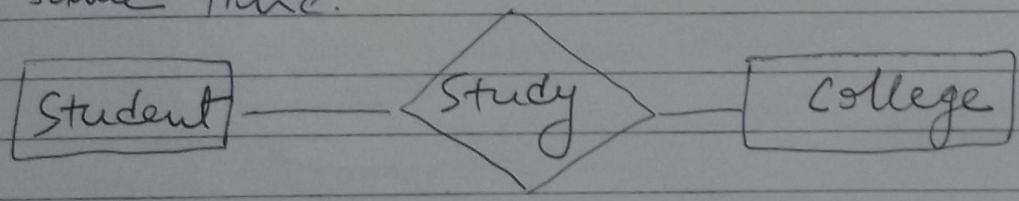


One-to-one relationship

2. One-to-many: When a single instance of an entity is associated with more than one instances of another entity then it is called one-to-many relationships.
for e.g → a client can place many orders; an order can not be placed by many customers.

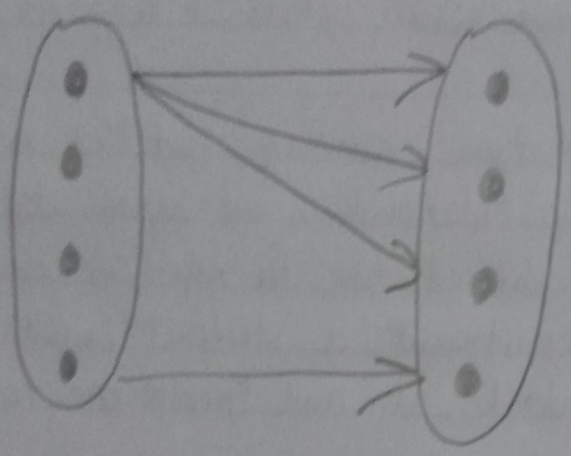


3. Many-to-One: More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.
for eg → many students can study in a single college, but a student cannot study in many colleges at the same time.



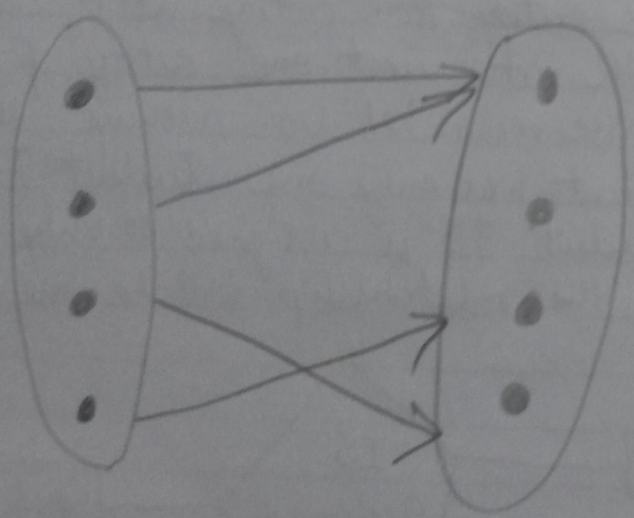
4. Many-to-Many: One entity from A can be associated with more than one entity from B & vice-versa. for example, the student can be assigned to many projects, & a project can be assigned to many

Using sets, it can be represented as:

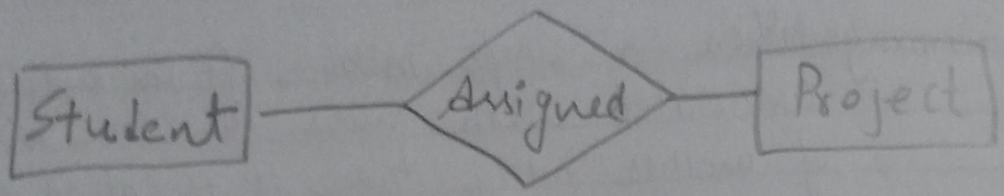


One-to-many relationship

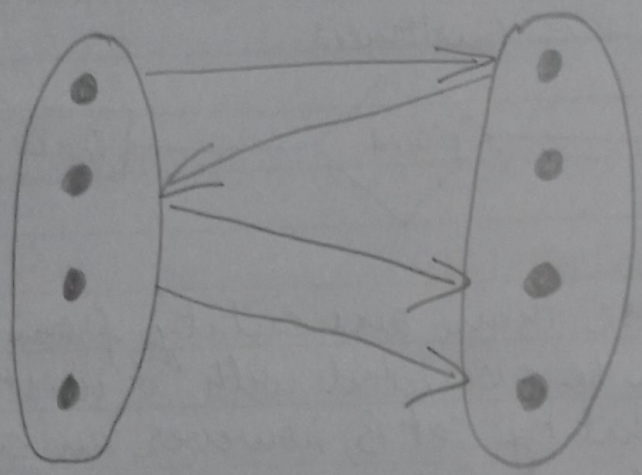
Using sets, it can be represented as:-



Many-to-one relationship

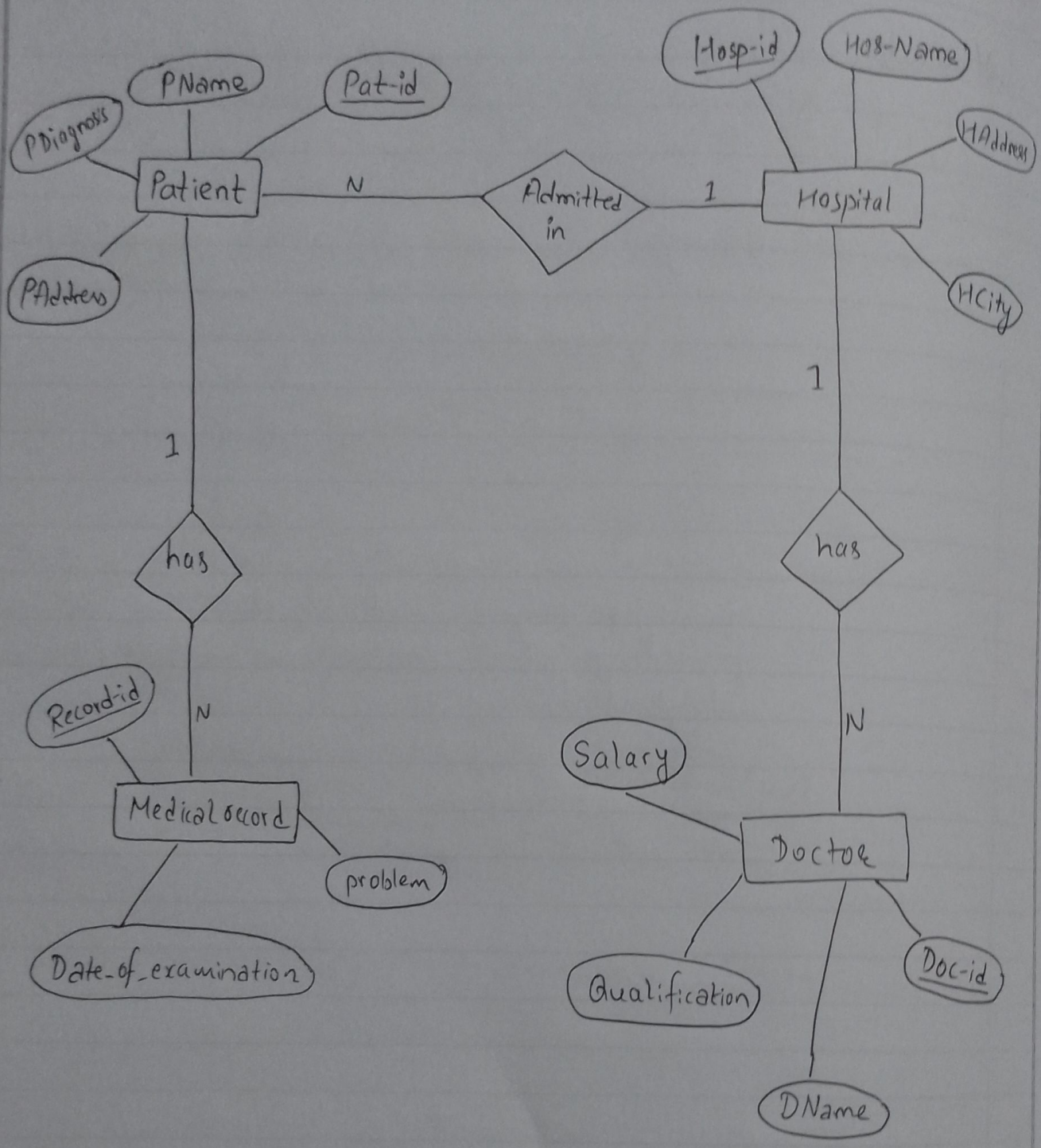


Using Sets, it can be represented as:-



Many - to - Many Relationship

E-R diagram of Hospital management System:-



Aim: To study about the class Diagram (with examples)

Class Diagram:
The class diagram depicts a static view of an application. It represents the type of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions & relationships to give an overview of the software system. It constitutes class names, attributes & functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams:

1. It analyse and design a static view of an application
2. It describe the major responsibilities of a system.
3. It is a base for component & deployment diagrams
4. It incorporate forward and reverse engineering.

Benefits of Class Diagrams:
1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better

understanding.

4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

Important components of Class Diagram:-

The class diagram is made up of three sections:-

- Upper Section: The uppe section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations & semantics. Some of the following rules that should be taken into account while representing a class are given below:-

  a. Capitalize the initial letter of the class name.
  b. Place the class name in the centre of the upper section.
  c. A class name must be written in bold format.
  d. The name of the abstract class should be written in italics format.

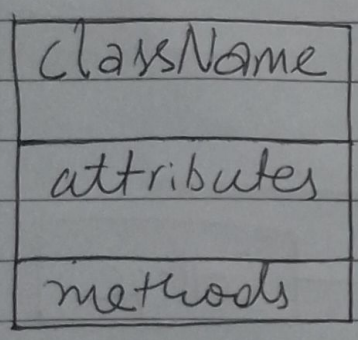
- Middle Section: The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:

  a. The attributes are written along with its visibility factors, which are public (+), private (−) protected (#) & package (~).

b. The accessibility of an attribute class is illustrated by its visibility factor.

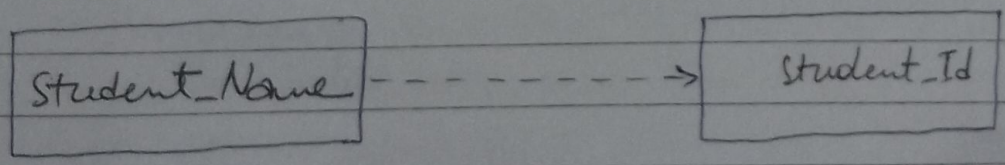
c. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

- Lower Section: The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



Types of relationships:

1. Dependency: It is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.
In the following example, Student_Name is dependent on the Student_Id.



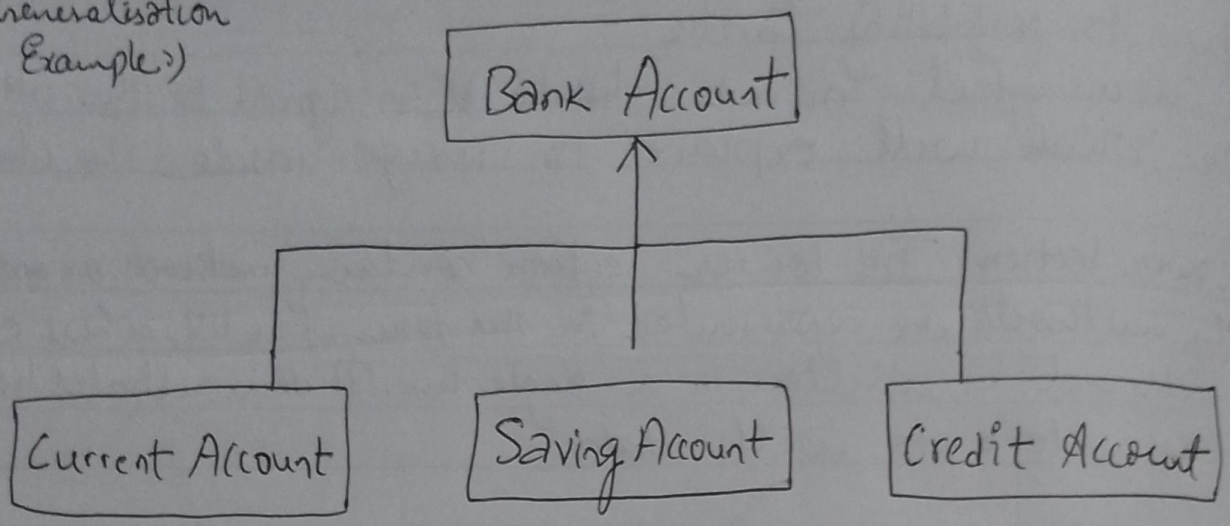
2. Generalisation: A generalisation is a relationship between a parent class (superclass) & a child class (subclass). In this, child class is inherited from the parent class.
for- eg. The Current Account, Saving Account, & Credit Account are the generalized form of Bank Account.

3. Association: It describes a static or physical connection between two or more objects. It depicts how many objects are there in relationship.

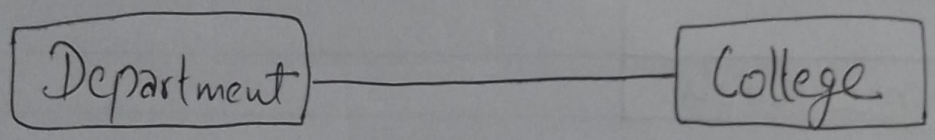
4. Aggregation: It is a subset of association, which represents a relationship. It is more specific than association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class ~~is~~ can exist independently of its parent class.
Eg. The company encompasses a number of employees, & even if one employee resigns, the company still exists

5. Composition: It is a subset of aggregation. It protrays the dependency b/w the parent & its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship. A contact book consist of multiple contacts, & if you delete the contact book,

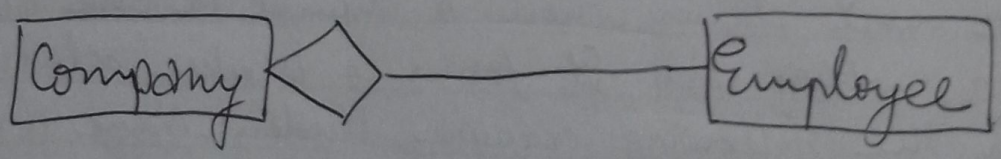
(Generalisation Example:)

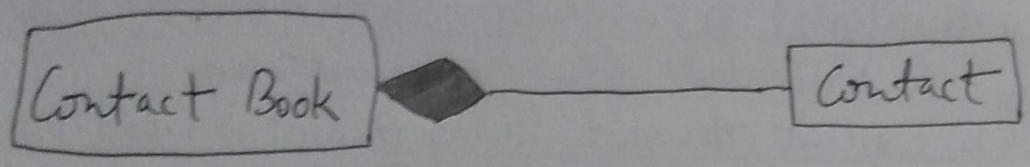


(Association Example)

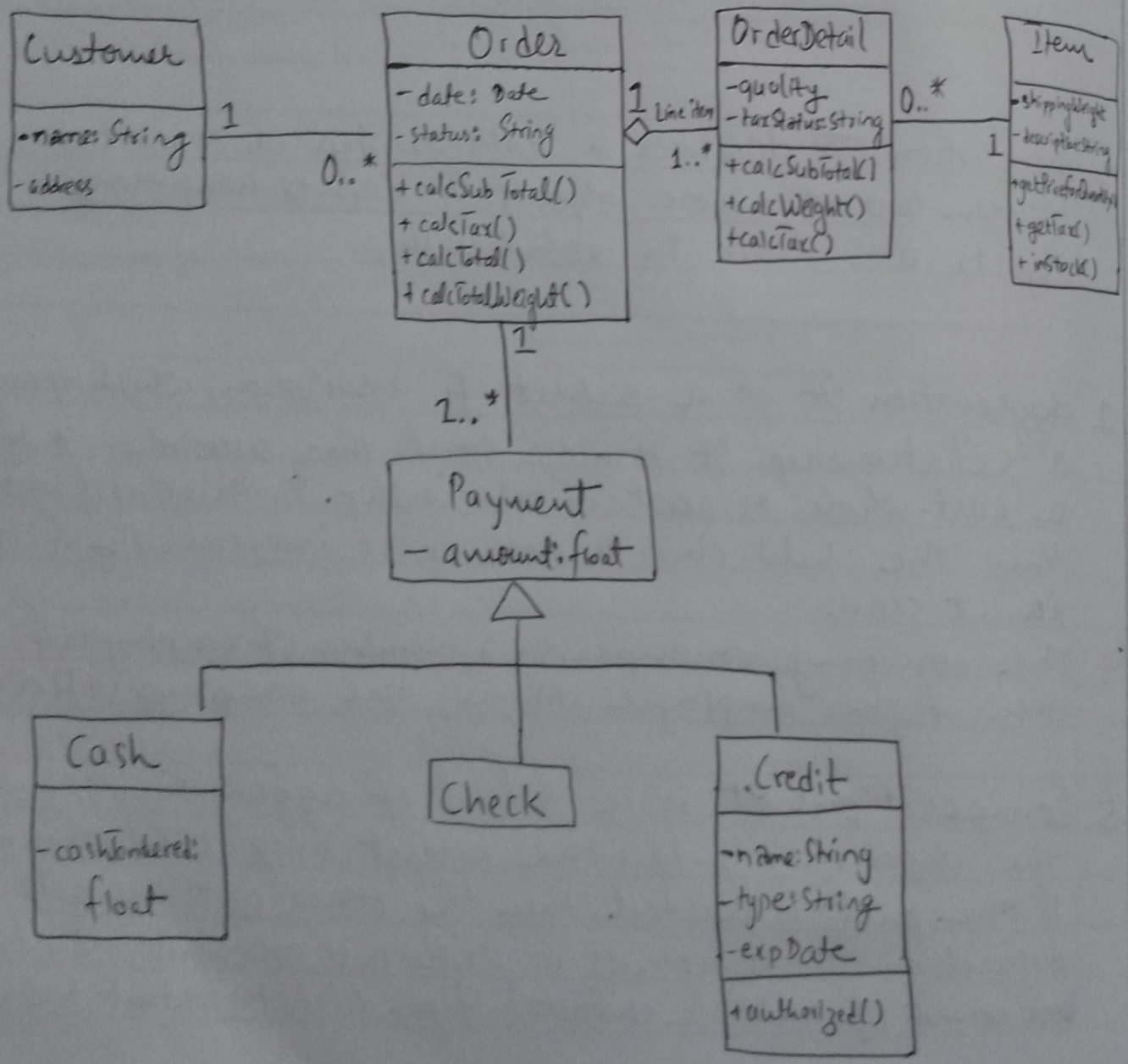


(Aggregation Example)





A class diagram describing sales order system is given below:-



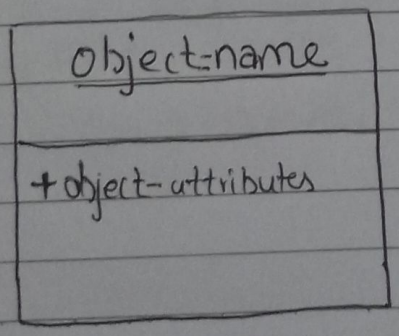
Name of the Experiment: To study about the Object Diagram (with example)

Object Diagram:-

Object Diagrams are dependent on the Class Diagrams as they are derived from class diagrams. It represents an instance of a class diagram. The objects help in portraying a static view of an object oriented system at a specific instant.

Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.

Notation of an Object Diagram:-



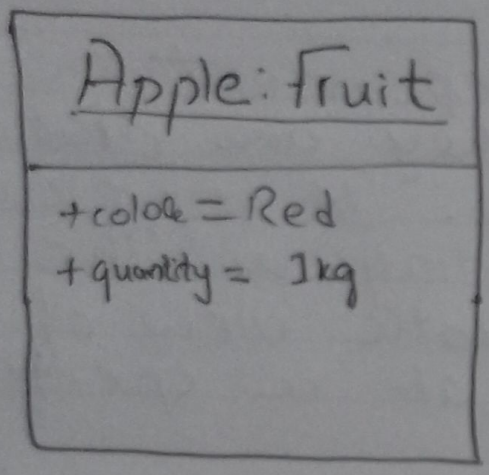
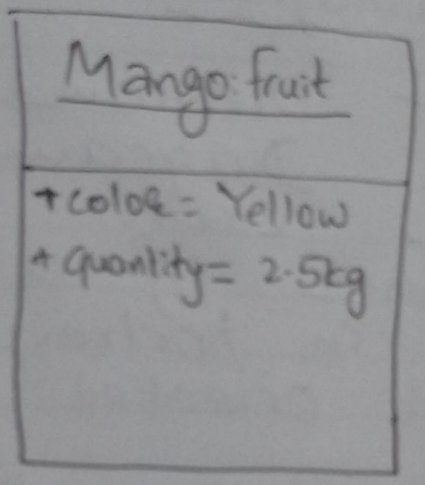
Purpose of Object - Diagram:-

The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes & their relationships whereas the object diagram represents an instance at a particular point of time.

Following are the purposes enlisted below:-

- It is used to perform forward & reverse engineering.
- It is used to understand object behaviour & their relationships practically.
- It is used to get a static view of a system.
- It is used to represent an instance of a system.

Applications Of objet diagram:

1. To build a prototype of a system.
2. To model complex data structures.
3. To perceive the system from a practical perspective.
4. Reverse engineering.

Example of Object Diagram:-