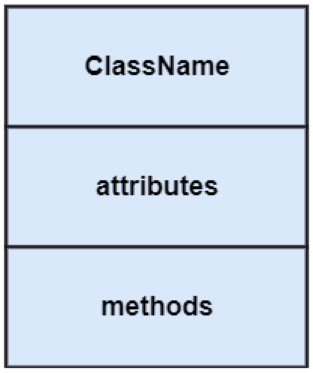
UML:

- UML stands for Unified Modelling Language.
- It is a standard language for specifying, visualizing, constructing, and documenting the artefacts (major elements) of software systems.
- It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95.

UML – Class Diagram:

- The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them.
- It shows the attributes, classes, functions, and relationships to give an overview of the software system.
- It constitutes class names, attributes, and 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**.

Vital components of Class Diagram are as follows:



- **Upper Section:**
  - The upper section encompasses the name of the class.
  - A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics.

- Some of the following rules that should be taken into account while representing a class are given below:
  - Capitalize the initial letter of the class name.
  - Place the class name in the centre of the upper section.
  - A class name must be written in bold format.
  - The name of the abstract class should be written in italics format.

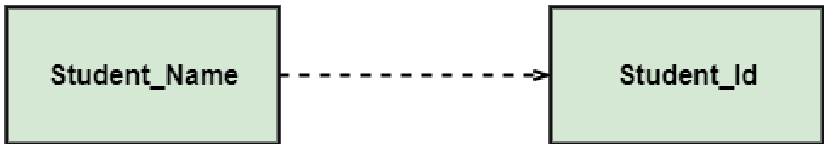
- **Middle Section:**
  - The middle section constitutes the attributes, which describe the quality of the class.
  - The attributes have the following characteristics:
    - The attributes are written along with its visibility factors, which are

public (+)	private (-)
protected (#)	package (~).
    - The accessibility of an attribute class is illustrated by the visibility factors.
    - A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

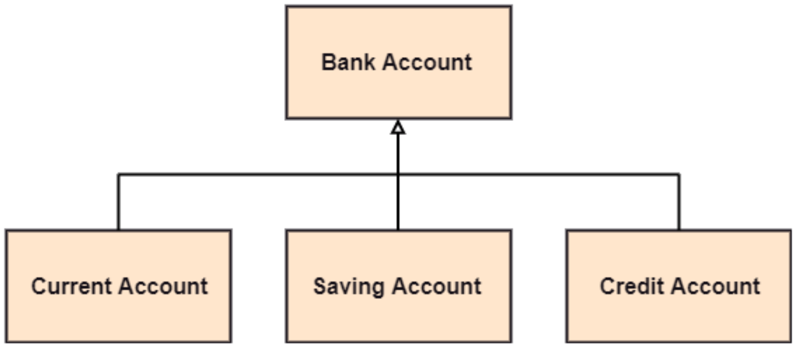
- **Lower Section:**
  - The lower section contains methods or operations.
  - The methods are represented in the form of a list, where each method is written in a single line.
  - It demonstrates how a class interacts with data.

Relationships in a Class Diagram are as follows:

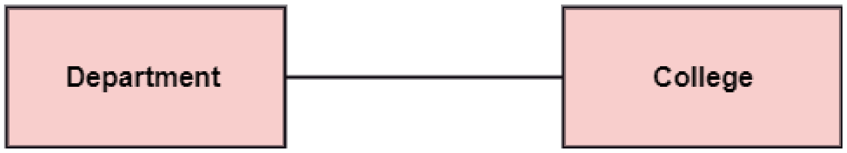
- **Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship. In the following example, Student_Name is dependent on the Student_Id.



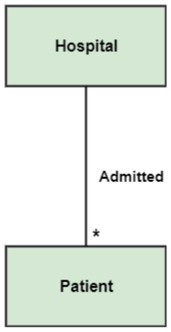
- **Generalization:** A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



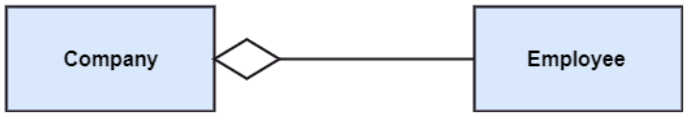
- **Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship. For example, a department is associated with the college.



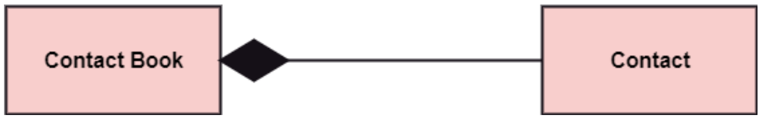
- **Multiplicity:** It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity. For example, multiple patients are admitted to one hospital. [1 ---- 0..* means 1 to zero or more multiplicity, 1 1..* means 1 to one or more multiplicity]



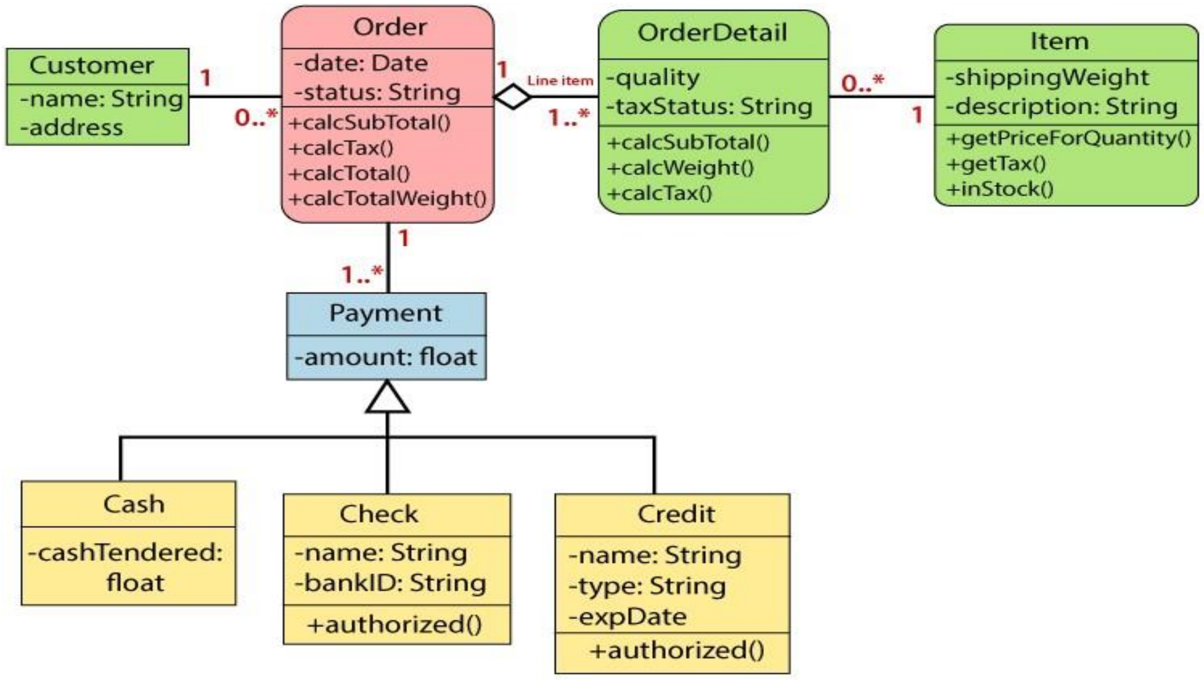
- **Aggregation:** An aggregation is a subset of association, which represents has a relationship. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class. The company encompasses a number of employees, and even if one employee resigns, the company still exists. Here parent Has-A relationship with child entity



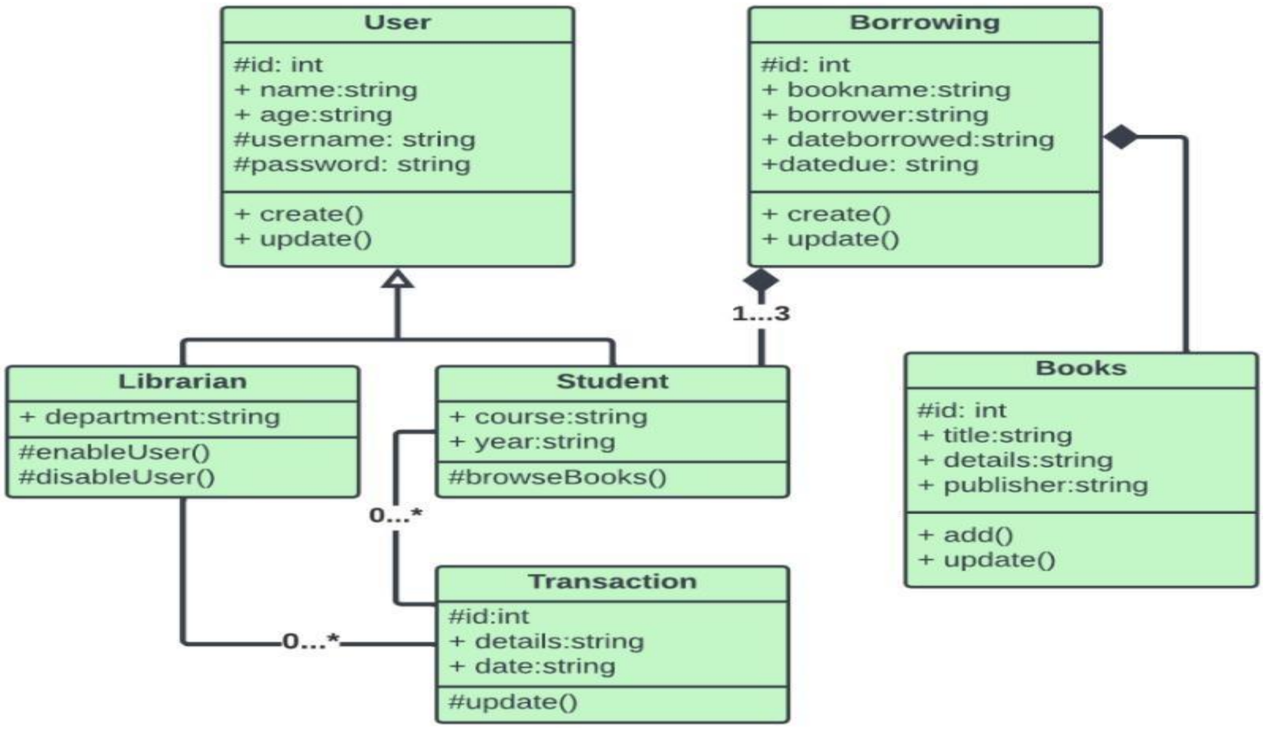
- **Composition:** The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship. A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost. Here parent entity owns child entity.



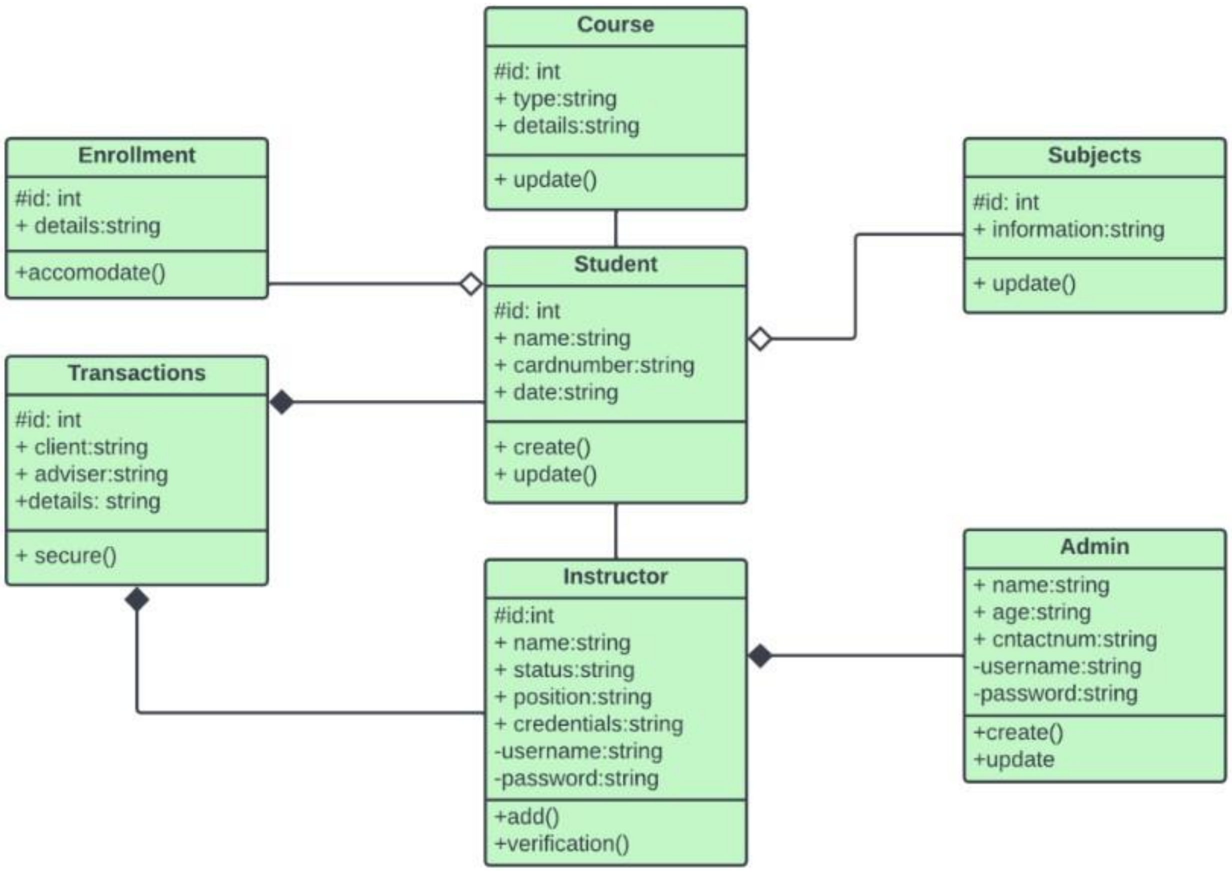
A class diagram describing the “sales order system” is given below:



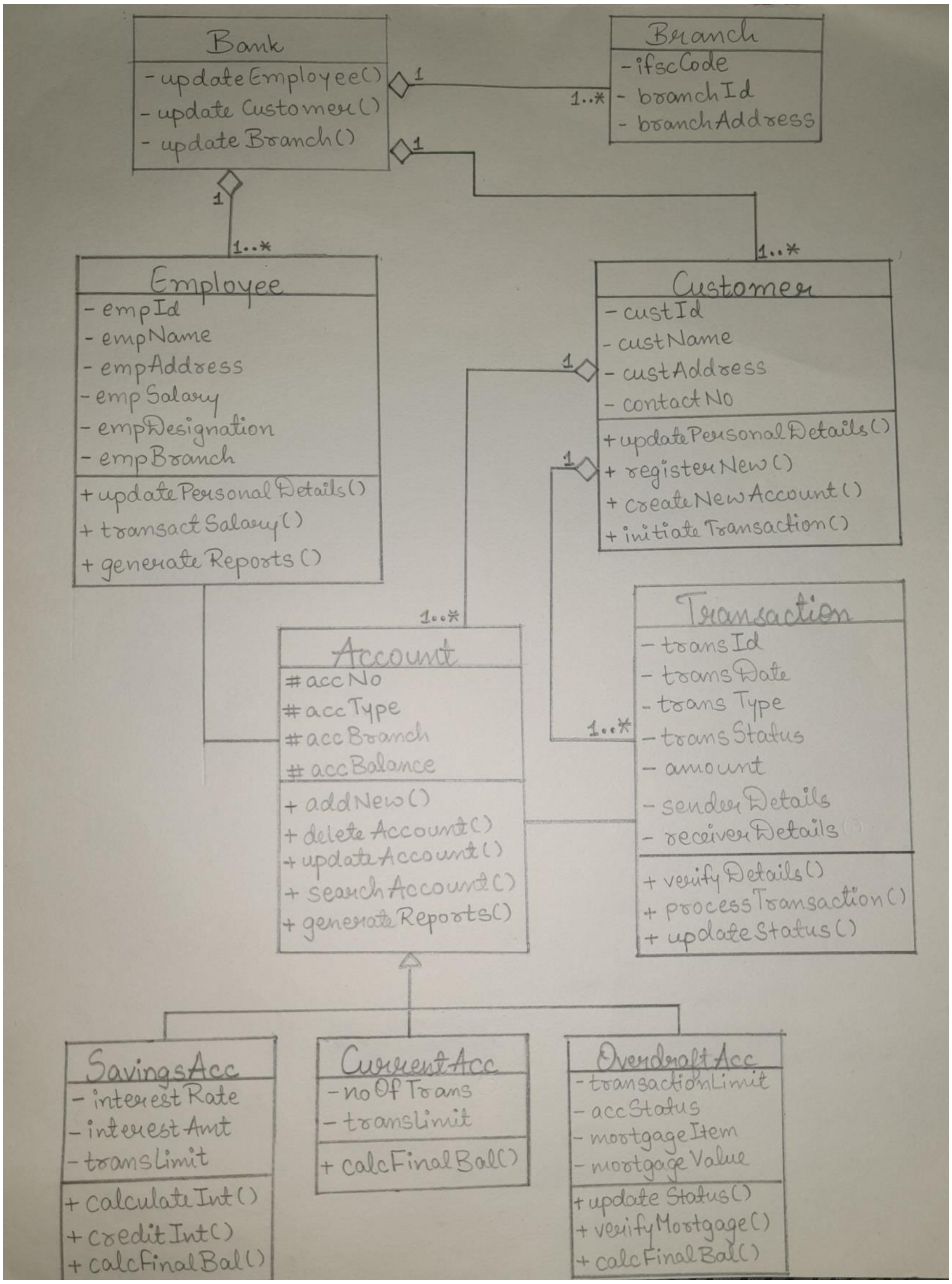
A class diagram describing the “library management system” is given below:



<u>A class diagram describing the “student management system” is given below</u>:



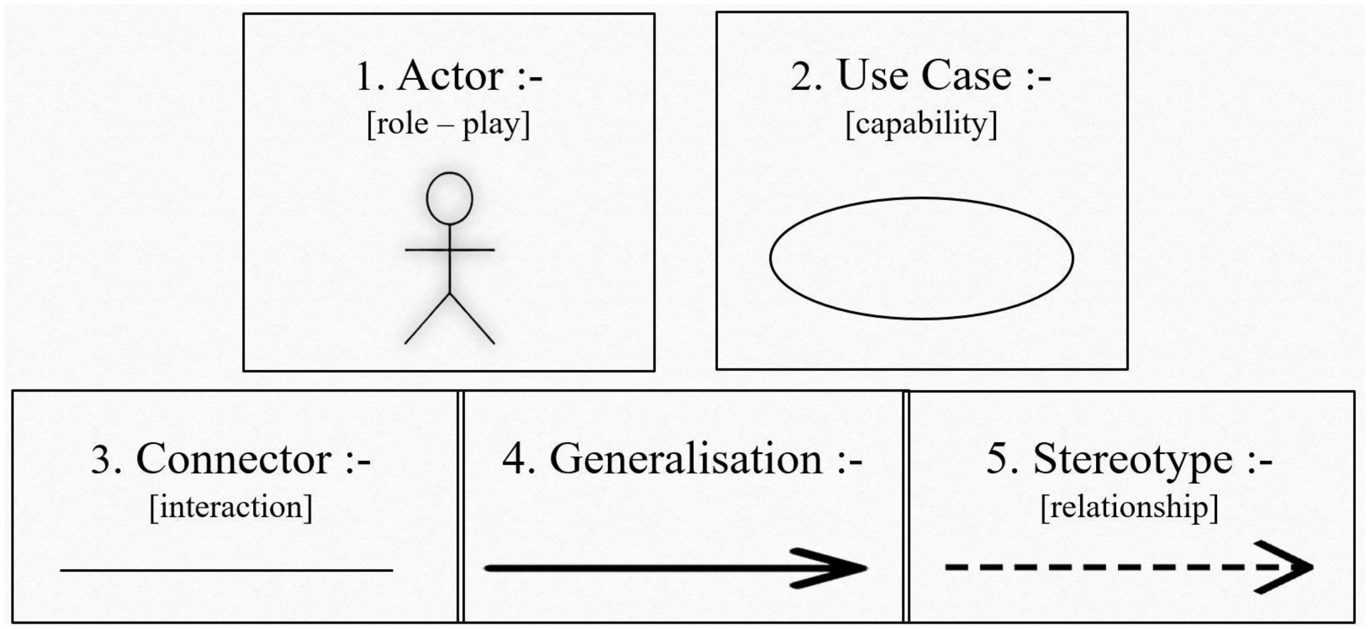
<u>A class diagram describing the “banking system” is given below</u>:



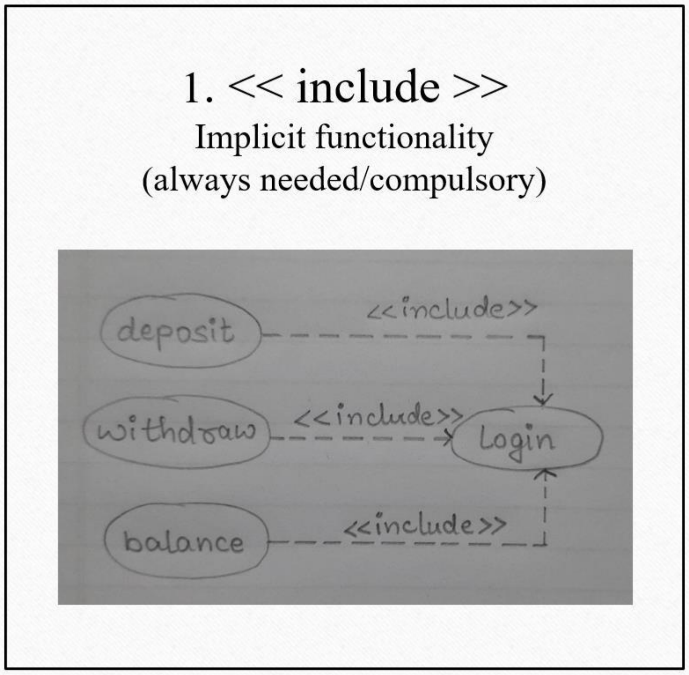
UML – Use Case Diagram:

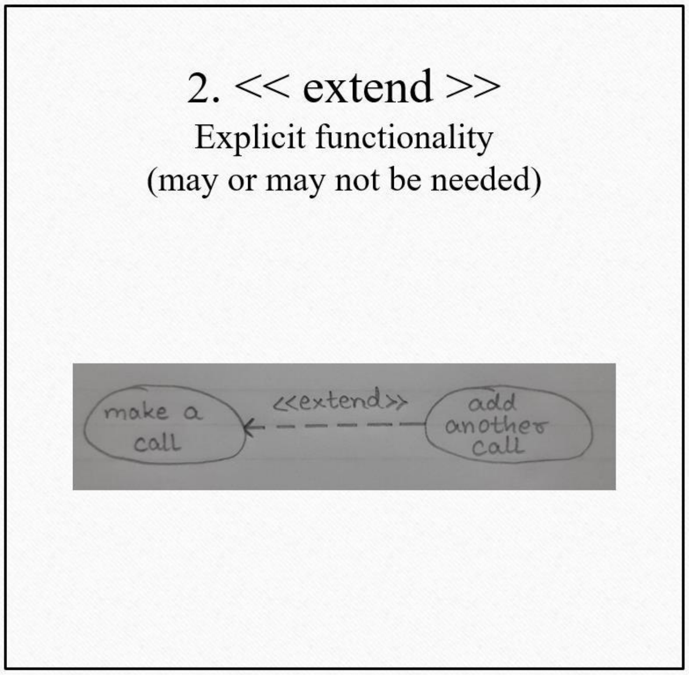
- A use case diagram is used to represent the dynamic behavior of a system.
- It encapsulates the system's functionality by incorporating use cases, actors, and their relationships.
- It models the tasks, services, and functions required by a system/subsystem of an application.
- Following are the purposes of a use case diagram given below:
  - It gathers the system's needs.
  - It depicts the external view of the system.
  - It recognizes the internal as well as external factors that influence the system.
  - It represents the interaction between the actors.

- Symbols used in use case diagram:

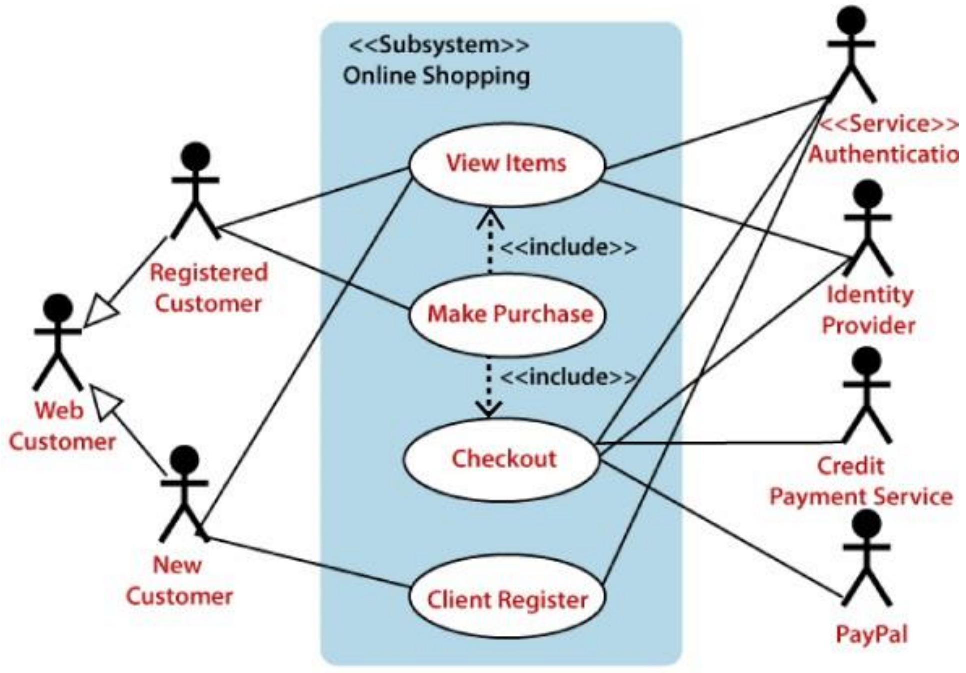


➢ Stereotypes (relationship) in use-case diagram:

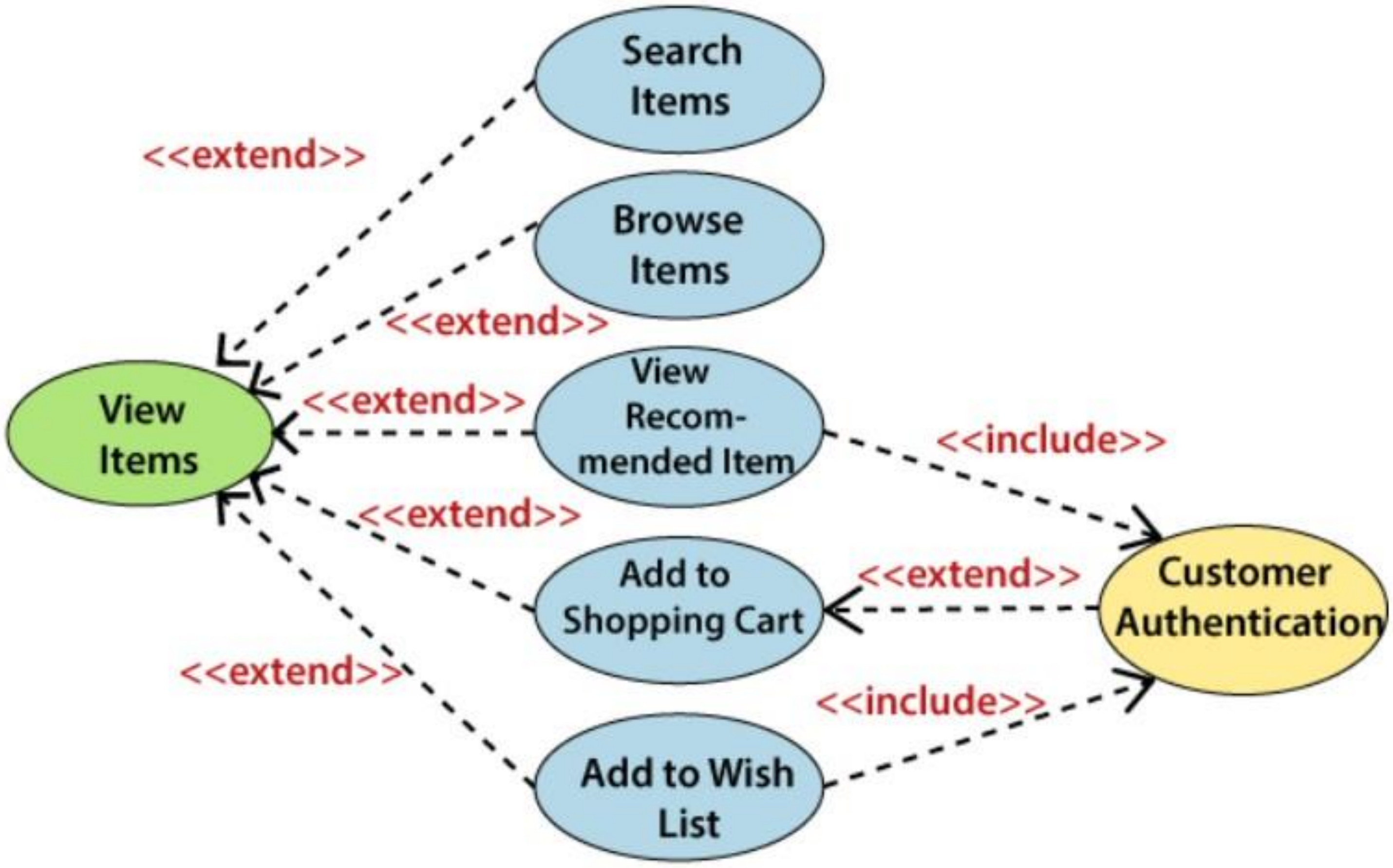

1. << include >>
Implicit functionality
(always needed/compulsory)


2. << extend >>
Explicit functionality
(may or may not be needed)

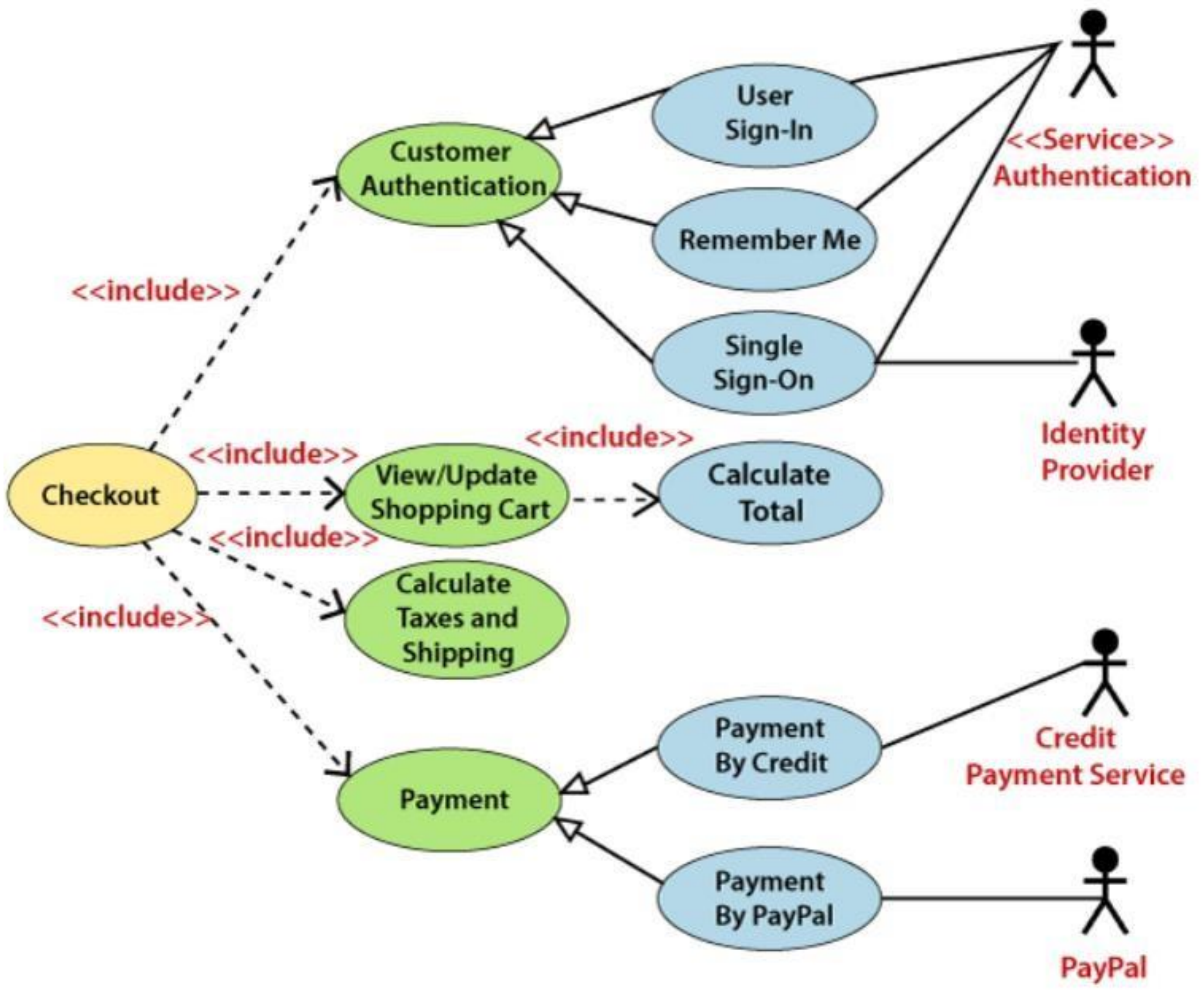
A use case diagram depicting the Online Shopping website is given below: [overall]



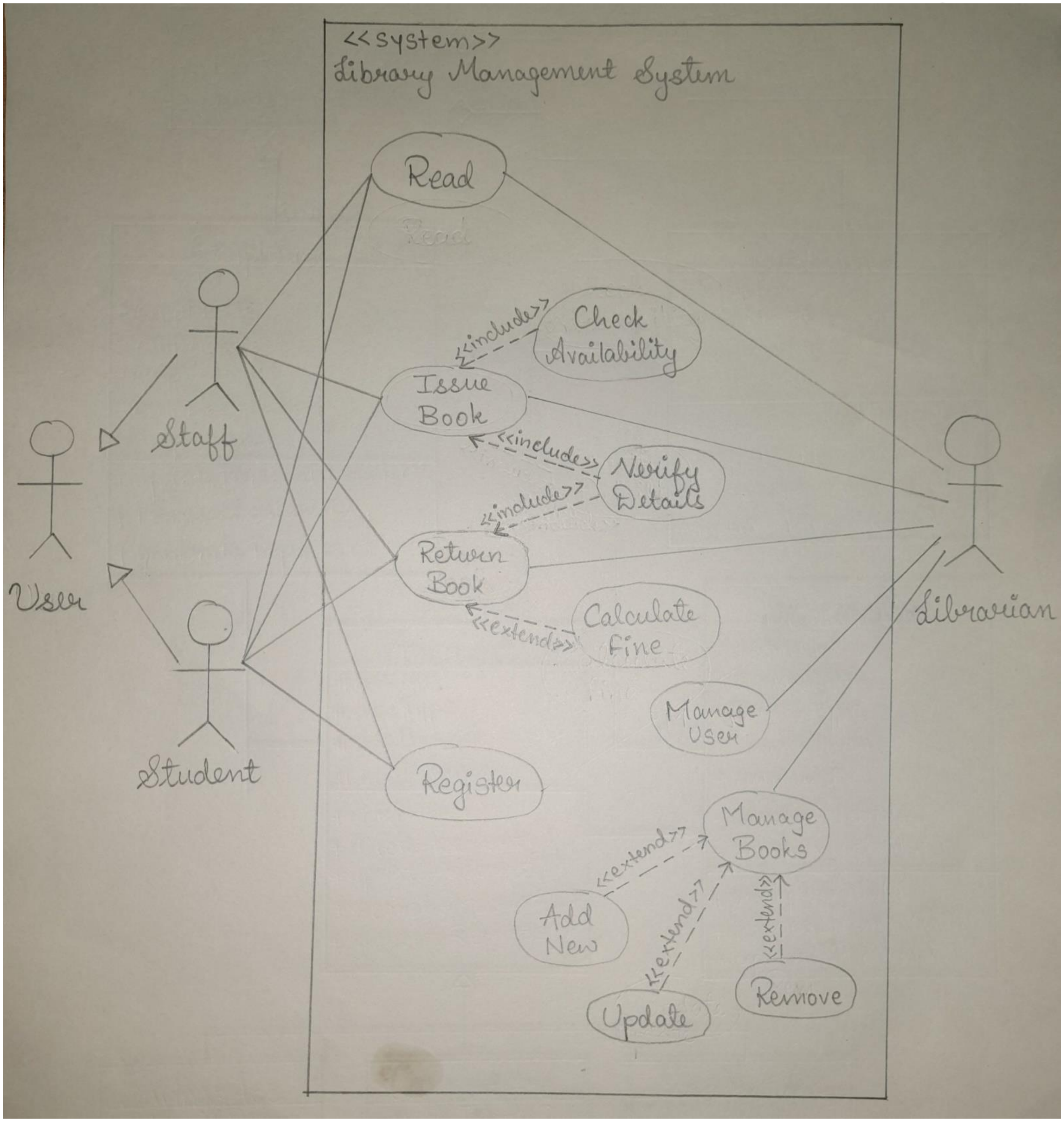
A use case diagram depicting the Online Shopping website is given below: [viewing items]



A use case diagram depicting the Online Shopping website is given below: [checkout]

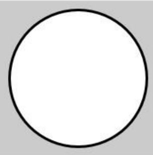
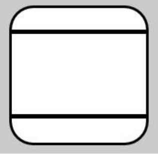
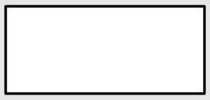
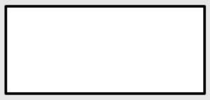
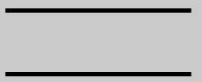
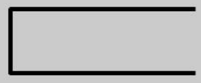
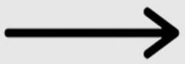
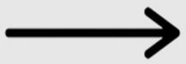


<u>A use case diagram depicting the Library Management System is given below</u>:



Data Flow Diagram:

- Data flow diagram is a graphical representation of defining system inputs, processorsoutput.
- It represents the flow of the data through the system.
- The DFDs are used in modern methods of system analysis.
- They are simple to the extent that the types of symbols and rules are very few.
- The DFDs serve two purposes
- Provided graphic tool which can be used by the analyst to explain his understanding of the system to the user.
- They can be readily converted into a structure chart which can be used in designing.
- The use of DFDs as a modeling tools was popularized by D Marco in 1978 and gain in 1979 through the structure system analysis methodology.

Sr. No.	Symbol Name	Description	Yourdon	Gane & Sarson
1.	Process	Here flow of data is transformed. Ex: verifying, updating, etc.		
2.	External Entity	A source or destination of data which is external to the system. Ex: supplier, customer, etc.		
3.	Data Store	Any stored data but with no reference to the physical method of storing. Ex: customer master, stock master, etc.		
4.	Dara Flow	It is a packet of data. It may be in the form of a document, letter, etc.		

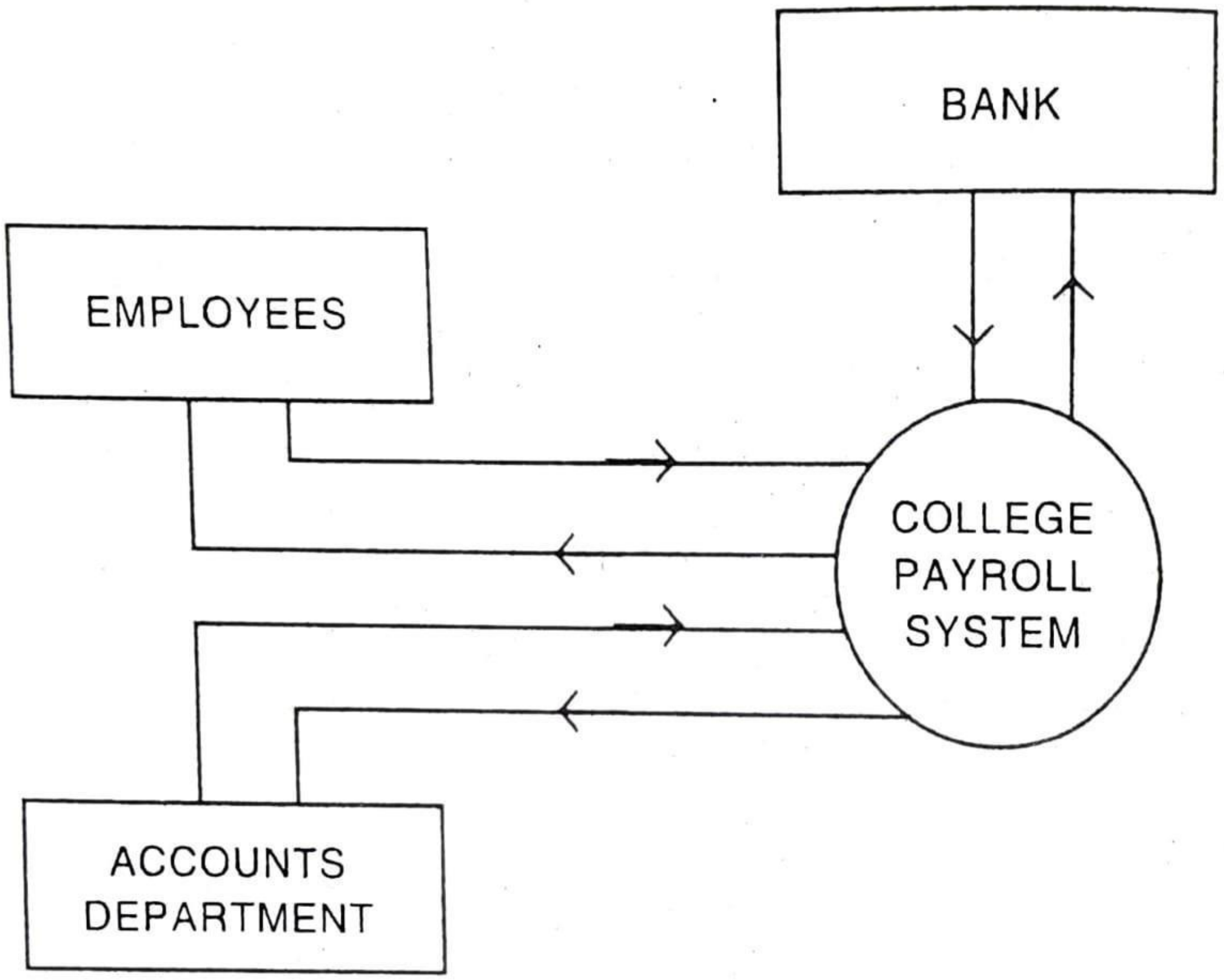


Fig. I Context diagram for the College Payroll System.

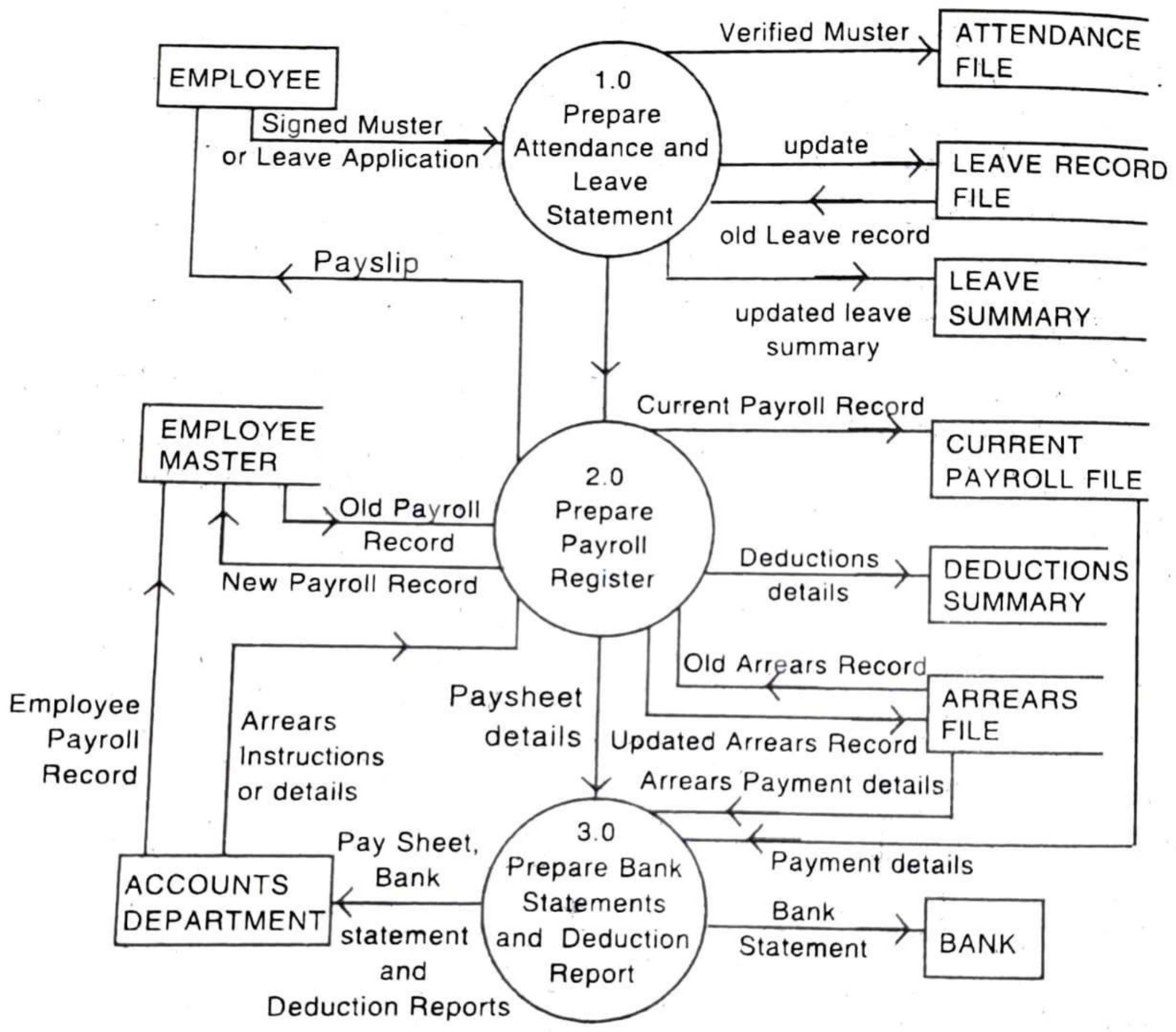


Fig. III Zero level DFD for College Payroll System.

First level DFD for College Payroll System :

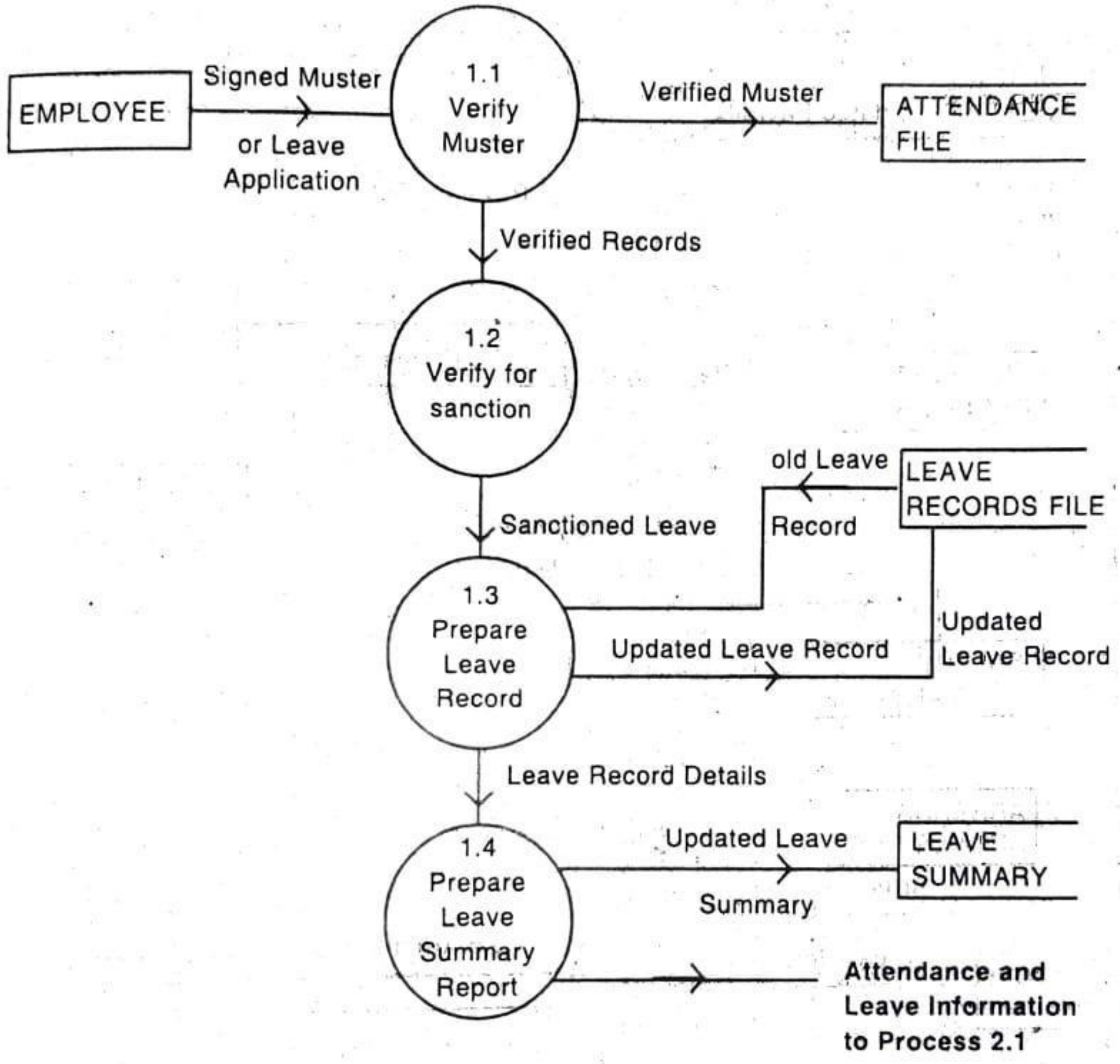


Fig. IV Level - 1 DFD for process 1.0

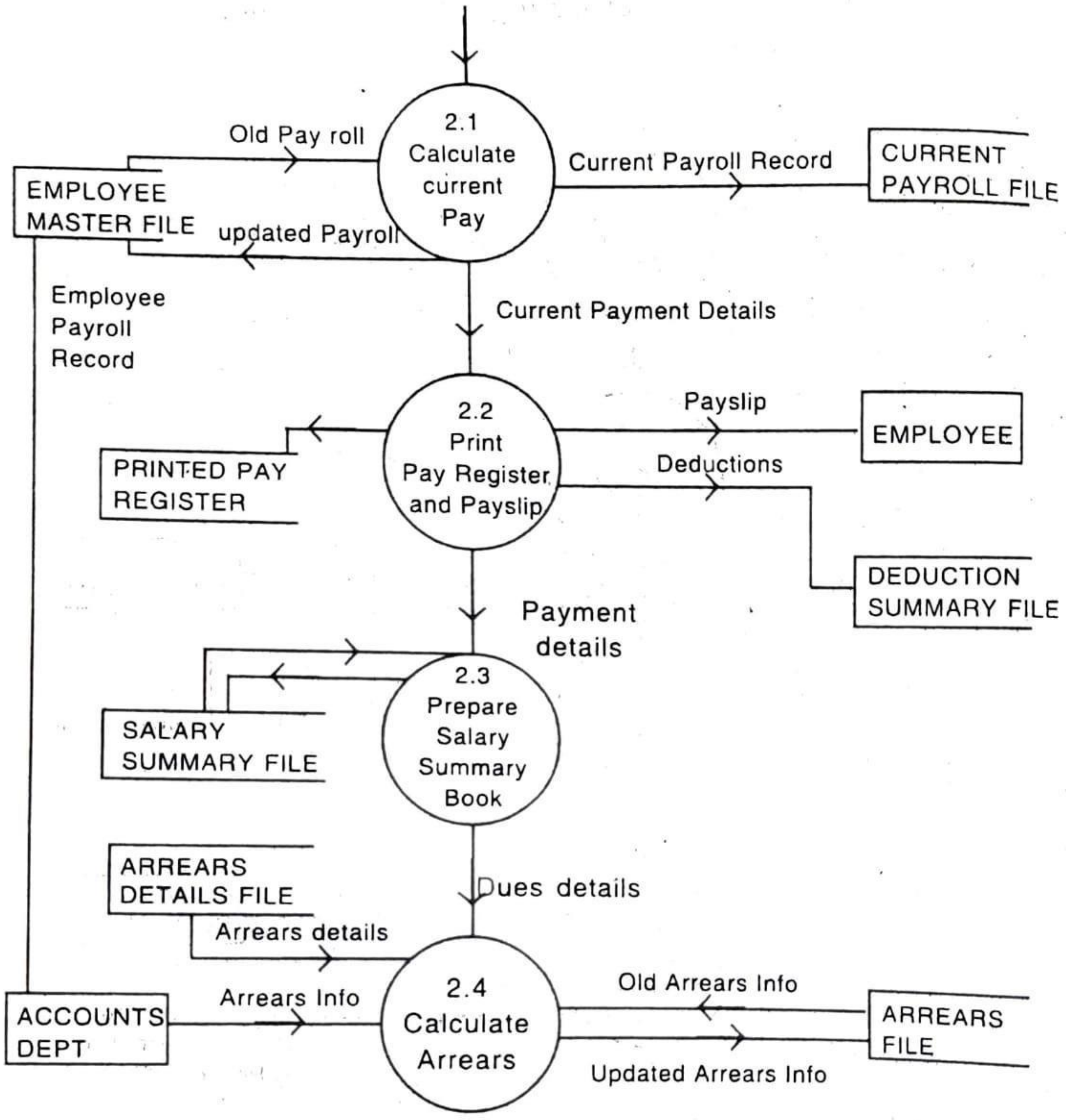


Fig. V Level - 1 DFD for process 2.0

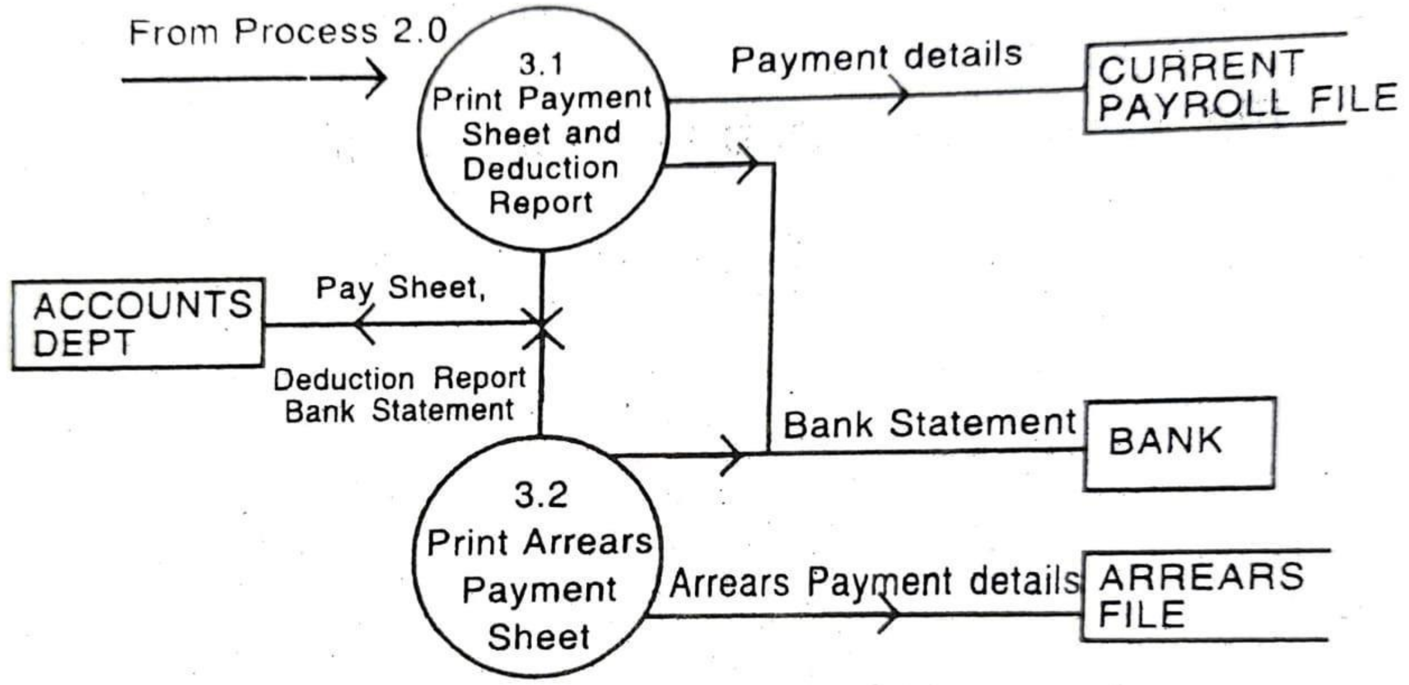


Fig. VI Level - 1 DFD for process 3.0

Data Dictionary:

- A data dictionary lists all data items appearing in the data flow diagram.
- Data dictionary is a set of meta-data which contains the definition and representation of data elements that are related to system.
- It stores data elements with accurate definition so that both user and system analyst will have a common understanding of input, output and component of stores.
- Data dictionary is the centralized collection of information about data.
- It stores meaning and origin of data, its relationship with other data, data format for usage etc.
- Data dictionary has rigorous definitions of all names in order to facilitate user and software designers.

<u>Format of data dictionary</u>:

Data Name	Name of the data elements/data flows/data stores.
Data aliases	An alternative name use for the connivance by multiple uses.
Data description	A short description of the data
Data stores	Name of the data store where data are going to store.
Format	Specify information about data types.
Physical location of data	In terms of record files or data base.
Data characteristics	Data length, range of data values or so on.

<u>Example of data dictionary</u>:

- Data Dictionary for Data Element – Employee Name:

Data Elements	Employee name
Description	First name, middle name and last name
Type	Character
Length	45
Alias	emp_nm

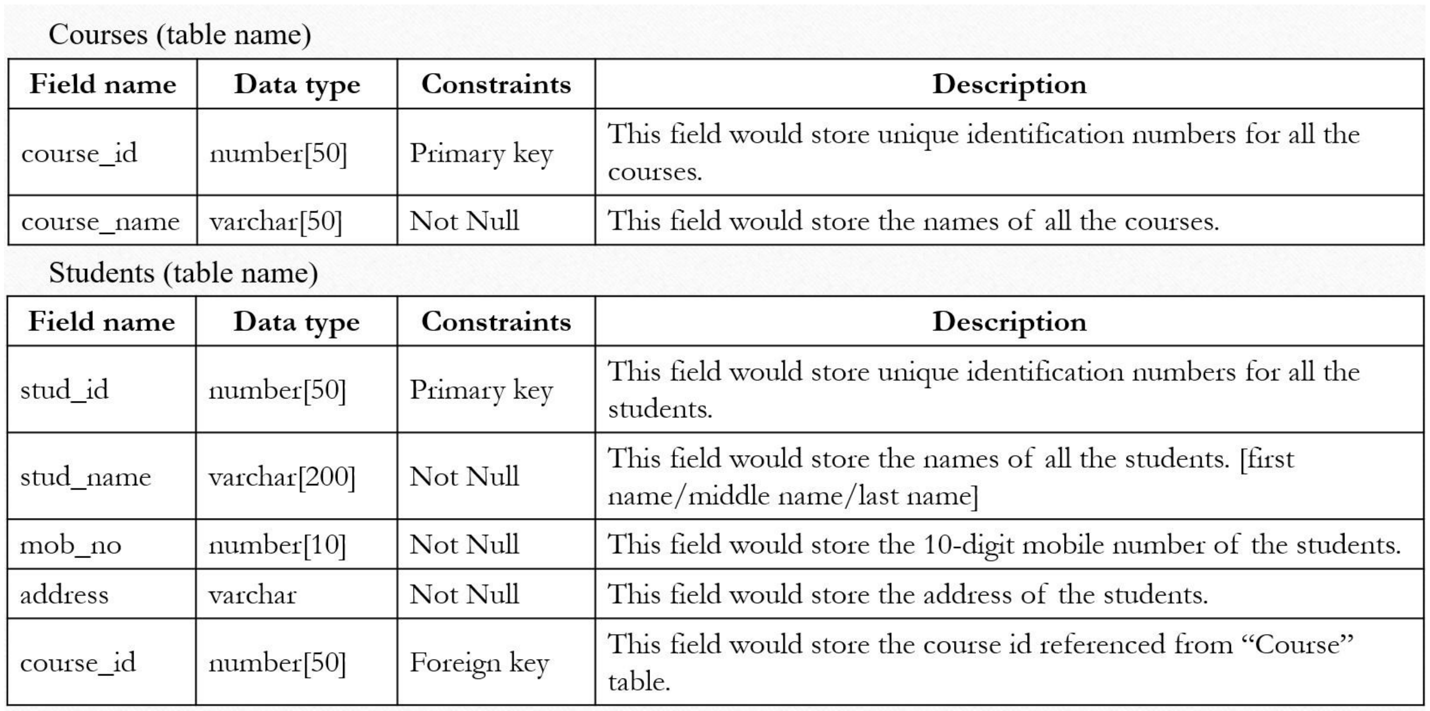
❖ Data Dictionary for Data Structure – Pay Slip:

Data Structure	Pay slip
Description	Gives the pay details of the employee for the month.
Content	emp_id, emp_name, grade, basic_pay, da
Where used	Process 2.2
Data flow	Print pay register and pay slips
Data store	Current payroll file

❖ Data Dictionary for Data Store – Arrears File:

Data Store	Arrears file
Description	Stores arrears/adjustment details.
Inbound data flow	Arrears due to promotion DA arrears Interim relief arrears
Outbound data flow	Arrear reports to admin

Another way to create data dictionary:



Courses (table name)

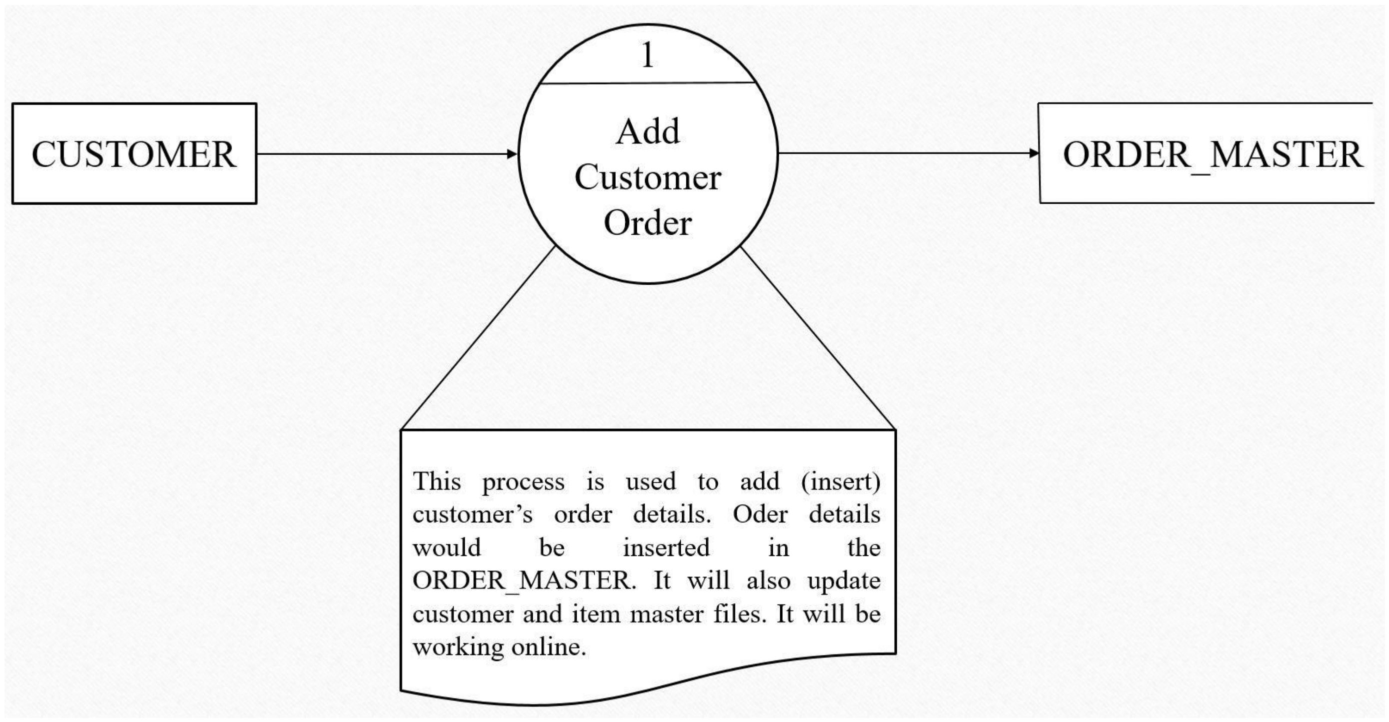
Field name	Data type	Constraints	Description
course_id	number[50]	Primary key	This field would store unique identification numbers for all the courses.
course_name	varchar[50]	Not Null	This field would store the names of all the courses.

Students (table name)

Field name	Data type	Constraints	Description
stud_id	number[50]	Primary key	This field would store unique identification numbers for all the students.
stud_name	varchar[200]	Not Null	This field would store the names of all the students. [first name/middle name/last name]
mob_no	number[10]	Not Null	This field would store the 10-digit mobile number of the students.
address	varchar	Not Null	This field would store the address of the students.
course_id	number[50]	Foreign key	This field would store the course id referenced from “Course” table.

Process Specification:

- Process Specification (PSPEC) can be used to specify the processing details implied by bubble within DFD.
- This specification describe the input to a function, the algorithm, also indicates restrictions and limitation imposed on the process.
- Process specification is used to describe the inner working of process representation in a flow.
- Goals to produce process specification:
    - Reduce process ambiguity.
    - Obtain a description of what is accomplished.
    - Validate the system design, including DFD and data dictionary.


1
Add Customer Order
CUSTOMER
ORDER_MASTER
This process is used to add (insert) customer’s order details. Oder details would be inserted in the ORDER_MASTER. It will also update customer and item master files. It will be working online.

Design Principles:

a) **Software design should correspond to the analysis model:** Often a design element corresponds to many requirements, therefore, we must know how the design model satisfies all the requirements represented by the analysis model.

b) **Choose the right programming paradigm:** A programming paradigm describes the structure of the software system. Depending on the nature and type of application, different programming paradigms such as procedure oriented, object-oriented, and prototyping paradigms can be used. The paradigm should be chosen keeping constraints in mind such as time, availability of resources and nature of user's requirements.

c) **Software design should be uniform and integrated:** Software design is considered uniform and integrated, if the interfaces are properly defined among the design components. For this, rules, format, and styles are established before the design team starts designing the software.

d) **Software design should be flexible:** Software design should be flexible enough to adapt changes easily. To achieve the flexibility, the basic design concepts such as abstraction, refinement, and modularity should be applied effectively.

e) **Software design should ensure minimal conceptual (semantic) errors:** The design team must ensure that major conceptual errors of design such as ambiguousness and inconsistency are addressed in advance before dealing with the syntactical errors present in the design model.

f) **Software design should be structured to degrade gently:** Software should be designed to handle unusual changes and circumstances, and if the need arises for termination, it must do so in a proper manner so that functionality of the software is not affected.

g) **Software design should represent correspondence between the software and real-world problem:** The software design should be structured in such a way that it always relates with the real-world problem.

h) **Software reuse:** Software engineers believe on the phrase: 'do not reinvent the wheel'. Therefore, software components should be designed in such a way that they can be effectively reused to increase the productivity.

i) **Designing for testability:** A common practice that has been followed is to keep the testing phase separate from the design and implementation phases. That is, first the software is developed (designed and implemented) and then handed over to the testers who subsequently determine whether the software is fit for distribution and subsequent use by the customer. However, it has become apparent that the process of separating testing is seriously flawed, as if any type of design or implementation errors are found after implementation, then the entire or a substantial part of the software requires to be redone. Thus, the test engineers should be involved from the initial stages. For example, they should be involved with analysts to prepare tests for determining whether the user requirements are being met.

j) **Prototyping:** Prototyping should be used when the requirements are not completely defined in the beginning. The user interacts with the developer to expand and refine the requirements as the development proceeds. Using prototyping, a quick 'mock-up' of the system can be developed. This mock-up can be used as a effective means to give the users a feel of what the system will look like and demonstrate functions that will be included in the developed system. Prototyping also helps in reducing risks of designing software that is not in accordance with the customer's requirements.

Design Concept:

1). Abstraction:

- It concentrates on the essential features and ignores details that are notrelevant. This means hiding the complexity.
- **Procedural Abstraction:**
  - It is name sequence of instructions that has a specific and limited function.
  - E.g. procedural abstraction would be the word open for a door for open implies a long sequence of procedural steps like walk to the door, reached out and knocked and pull door and step away from moving door etc.
  - Means in short hiding the Procedure details here hiding the procedure of how to open door.
- **Data Abstraction:**
  - Data abstraction is a name collection of data that describes a data object in the context of the procedural abstraction open we can define a data abstraction called door like any data object.
  - The data abstraction for door would be a set of attributes that describe the door.
  - E.g. Door type, swing direction, opening mechanism, weight dimension etc. it follows that the procedural abstraction would make use of information contain in the attributes of the data abstraction door.
  - Means in short hiding the data details here hide the door details.
- **Control Abstraction:**
  - Implies a program control mechanisms without specify internal details.
  - Means in short hiding the Control Details.

2). Refinements:

- The process of program refinement (modification or enhancement) is a partitioning process i.e. used during requirement analysis.
- Refinement is actually a process of elaboration (expansion). we begin with a statement of function or description of information i.e. defines a high level of abstraction.
- The statement describes function or information conceptually but provides no information about the internal working of the functions or the internal structure of the information.
- Refinement causes the system designer to elaborate on the original statement providing more and more detail as each successive refinement occurs.
- Abstraction and refinement are complimentary concepts. Abstractions enable a designer to specify procedure and data refinement helps the designer a detail at a low level.

3). Modularity:

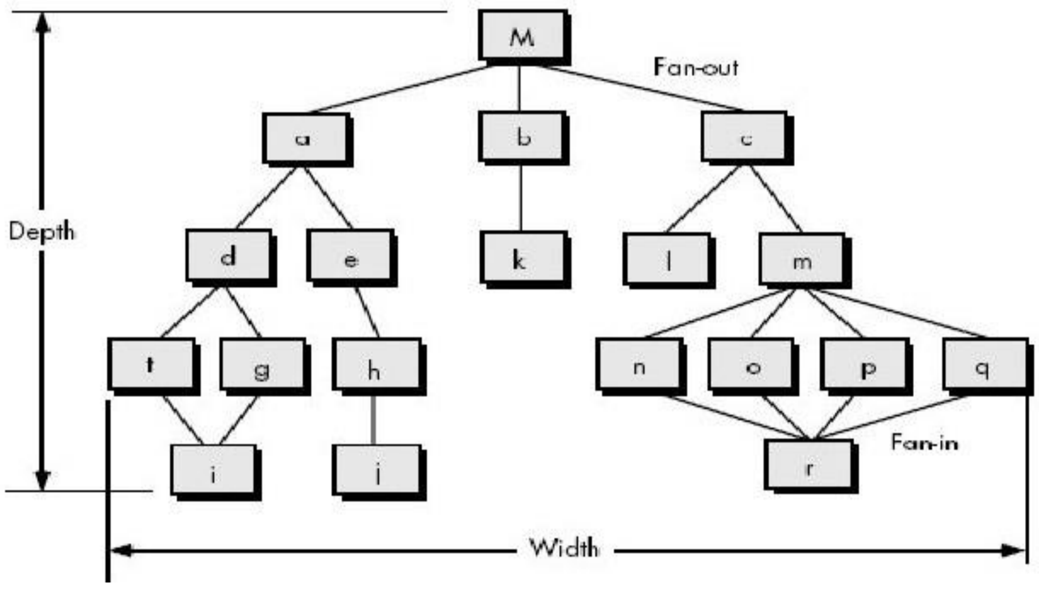
- Software is divided into separately named and addressable components which are called modules.
- Those are integrated to satisfy problem requirements.
- Modularity is a single attribute of software that allows a program to be intellectually manageable.
- Monolithic software i.e. a large program composed of a single module cannot be easily readable the number of control paths, spend of referential number of variables and overall complexity would make understanding close of impossible.

4). Software Architecture:

- Architecture is the hierarchical structure of program components or modules.
- There are some set of properties that should be specified as an architectural design.

5). Control Hierarchy or Program Module:

- The control hierarchy also called program structure represents the organization of program components and implies a hierarchy of control. It does not represent procedural aspects of software such as sequence of software, occurrence, and order of decision or representation of operations.
- In the given picture, depth and width provide an indication of the number of levels of control and overall span of control.
- Fan – out is a major of the number of modules that are directly controlled by another module.
- Fan – in indicate how many modules directly control a given module?
- The control relationship among modules is expressed in the following way:
    - A module that controls another module is said to be super ordinate to it and a module controlled by another is said to subordinate to the controller.
- In above figure module M is super-ordinate to modules a, b and c and module k is subordinate to module c.



6). Data Structure:

- Data structure is a representation of logical relationship among individual elements of data. Data structure is an important program structure to the representation of software architecture.
- Data structure shows the organization of methods to access a scalar item in the simplest form of all data structure. A scalar item represents a single element of information which is addressed by an identifier and i.e. accessed by specific a single address in memory.
- When scalar items are organized as a list of continuous group then a sequential vector is formed, when the sequential vector is extended into two or three or an arbitrary number of dimension then a ‘n’ dimension space is created and the most common ‘n’ dimension space is two dimensional matrix and ‘n’ dimension space is also called an array and a link list is a data structure that organized the memory elements into a non-contiguous scalar item or vector.

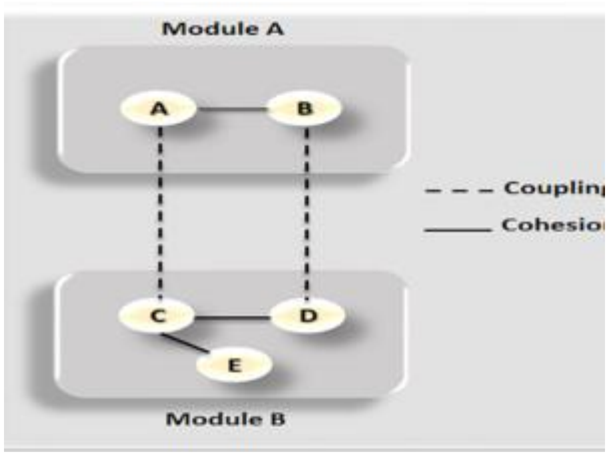
7). Software Procedure:

- Focus on the processing details of each module.
- Procedure must provide a exact specification of processing, including sequence of events, exact decision points, repetitive operation and even data organization and structure, there is relationship between structure and procedure.

What is functional independence in software engineering?

Functional independence in software engineering means that when a module focuses on a single task, it should be able to accomplish it with very little interaction with other modules.

In software engineering, if a module is functionally independent of other module then it means it has high cohesion and low coupling.



Example of functional independence

- We will take an example of simple college level project to *explain concept of functional independence*.
- Suppose you and your friends are asked to work on a calculator project as a team. Here we need to develop each calculator functionality in form of modules taking two user inputs.
- So our modules are additionModule, subtractionModule, divisionModule, multiplicationModule. Each one of you pick up one module for development purpose.
- Before you enter into development phase, you and your team needs to make sure to design the project in such a way that each of the module that you develop individually should be able to perform its assigned task without requiring much or no interaction with your friends module. What I intend to say is, if you are working on additionModule then your module should be able to independently perform addition operation on recieving user input. It should not require to make any

interaction with other modules like subtractionModule, multiplicationModule or etc.

- This is actually the concept of having module as *functional independence* of other modules.

<u>Effective modular design (Functional Independence) - COHESION</u>:

- Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component.

- Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.
- There are seven types of cohesion, namely –
- **Co-incidental cohesion:**
  It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not- accepted.
- **Logical cohesion:**
  When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion:**
  When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion:**
  When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion:**
  When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion:**
  When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion:**
  It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Effective modular design (Functional Independence) - COUPLING:

- Coupling is the measure of the degree of interdependence between the modules.
- A good software will have low coupling.
- There are five levels of coupling, namely:
- **Content coupling:**
  When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling:**
  When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling:**
  Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling:**
  When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling:**
  Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.