

# Requirements Engineering

---

Instructor: Mehroze Khan

# Requirements Engineering

- The **requirements** for a system are the descriptions of the **services** that a system should provide and the **constraints** on its operation.
- These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.
- The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering (RE)**.

# User Requirements

- User requirements are **statements**, in a natural language plus **diagrams**, of what services the system is expected to provide to system users and the constraints under which it must operate.
- The user requirements may vary from **broad statements** of the system features required to **detailed, precise descriptions** of the system functionality.



# System Requirements

- System requirements are more **detailed descriptions** of the software system's functions, services, and operational constraints.
- The **system requirements document** (sometimes called a functional specification) should define exactly what is to be implemented.



# Example

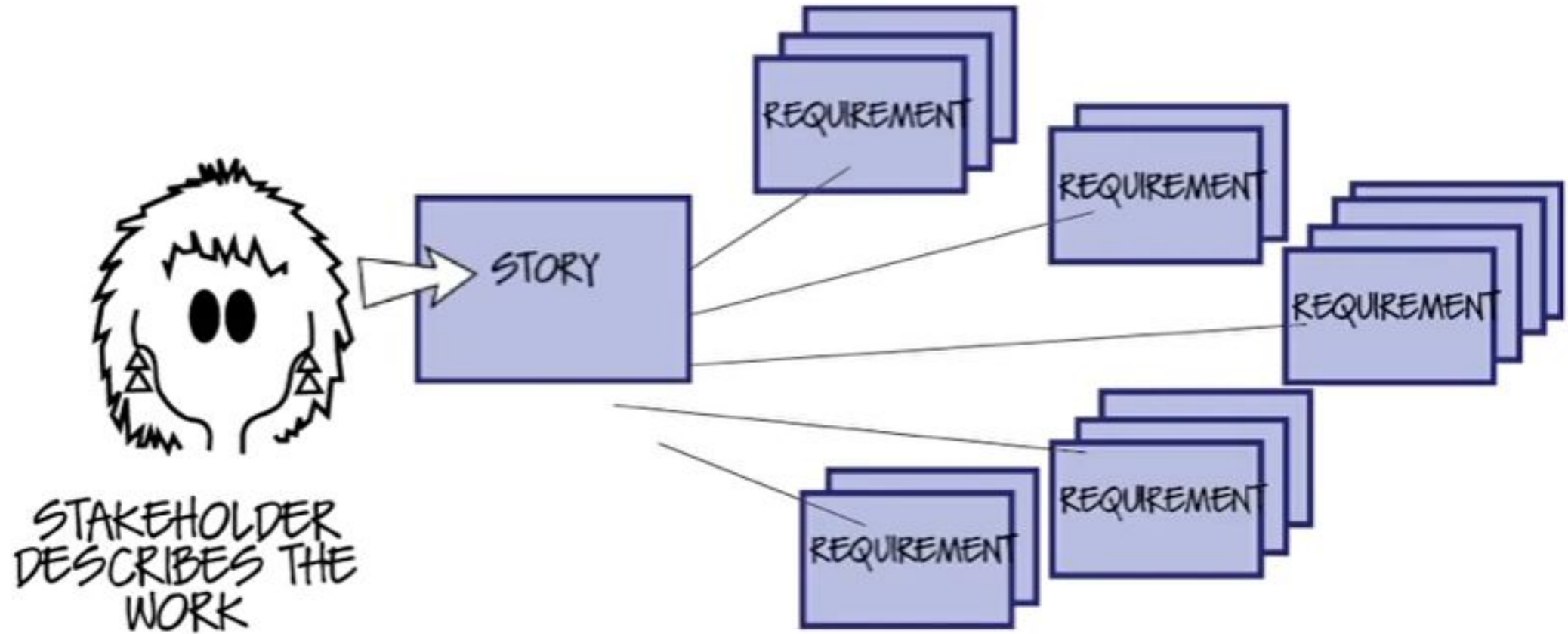
## User requirements definition

- 1.** The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

## System requirements specification

- 1.1** On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- 1.2** The system shall generate the report for printing after 17.30 on the last working day of the month.
- 1.3** A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
- 1.4** If drugs are available in different dose units (e.g. 10mg, 20mg, etc.) separate reports shall be created for each dose unit.
- 1.5** Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

# Requirement Engineering



# Requirements Engineering Tasks

Inception

Elicitation

Elaboration

Negotiation

Specification

Validation

Management



# Inception

- Identify: all stakeholders, measurable benefits of successful implementation, possible alternatives
- Ask questions stepwise, as early as possible, first meeting/encounter
- Possible questions at 1<sup>st</sup> step (stakeholders, overall goals and benefits):
  - Who is behind the request for this work?
  - Who will use this solution?
  - What will be the economic benefit of a successful solution?





# Inception

- Possible questions at 2<sup>nd</sup> step (detailed understanding and customer perception about the solution):
  - What problems(s) will this solution address?
  - Can you show me (or describe) the business environment in which the solution will be used?
  - How do you characterize the 'good' output?
- Possible questions at 3<sup>rd</sup> step (effectiveness of communication):
  - Are you the right person to answer these questions?
  - Are my questions relevant to the problem that you have?
  - Can anyone else provide additional information?



# Elicitation

- Get more detailed requirements.
- Customers do not always understand what their needs and problems are.
- It is important to discuss the requirements with everyone who has a stake in the system.
- Come up with agreement on what the requirements are
  - If we cannot agree on what the requirements are, then the project is doomed to fail

# Functional and Non-functional Requirements

- Software system requirements are often classified as functional or non-functional requirements:

## **1. *Functional requirements***

- These are statements of services the system should provide
- How the system should react to particular inputs
- How the system should behave in particular situations
- It may also explicitly state what the system should not do

## **2. *Non-functional requirements***

- These are constraints on the services or functions offered by the system
- Include timing constraints, constraints on the development process, and constraints imposed by standards

# Functional and Non-functional Requirements

- To put it simply, functional requirements describe **what the product should do**, while non-functional requirements place constraints on **how the product should do it**.
- They can be expressed in the following form:
  - Functional requirement:** "The system must do [requirement]."
  - Non-functional requirement:** "The system shall be [requirement]."
- **Example:**
  - Functional requirement:** "The system must allow the user to submit feedback through a contact form in the app."
  - Non-functional requirement:** "When the submit button is pressed, the confirmation screen shall load within 2 seconds."

# Functional Requirements

## **Functional Requirements**

### Functionality

- What will the system do?
- When will the system do it?
- Are there several modes of operation?
- What kinds of computations or data transformations must be performed?
- What are the appropriate reactions to possible stimuli?

### Data

- For both input and output, what should be the format of the data?
- Must any data be retained for any period of time?

# Example

- **User story:** As an existing user, I want to be able to log into my account.
- **Functional requirement:**
  - Log In
    - The system must allow users to log into their account by entering their email and password.
    - The system must allow users to log in with their Google accounts.
    - The system must allow users to reset their password by clicking on "I forgot my password" and receiving a link to their verified email address.

# Non-functional Requirements

- Non-functional or Quality requirements, as the name suggests, are requirements that are not directly concerned with the specific services delivered by the system to its users.
- These non-functional requirements usually specify or constrain characteristics of the system as a whole.
- A *nonfunctional requirement* (NFR) can be described as a quality attribute, a performance attribute, a security attribute, or a general constraint on a system.

# Example

- Security
  - Users shall be forced to change their password the next time they log in if they have not changed it within the length of time established as “password expiration duration”.
  - Passwords shall never be visible at the point of entry or at any other time.
  - Any new login shall be allowed after two step verification.



# Non-functional Requirements

## Quality Requirements

### Performance

- Are there constraints on execution speed, response time, or throughput?
- What efficiency measures will apply to resource usage and response time?
- How much data will flow through the system?
- How often will data be received or sent?

### Usability and Human Factors

- What kind of training will be required for each type of user?
- How easy should it be for a user to understand and use the system?
- How difficult should it be for a user to misuse the system?

### Security

- Must access to the system or information be controlled?
- Should each user's data be isolated from the data of other users?
- Should user programs be isolated from other programs and from the operating system?
- Should precautions be taken against theft or vandalism?

## Reliability and Availability

- Must the system detect and isolate faults?
- What is the prescribed mean time between failures?
- Is there a maximum time allowed for restarting the system after a failure?
- How often will the system be backed up?
- Must backup copies be stored at a different location?
- Should precautions be taken against fire or water damage?

## Maintainability

- Will maintenance merely correct errors, or will it also include improving the system?
- When and in what ways might the system be changed in the future?
- How easy should it be to add features to the system?
- How easy should it be to port the system from one platform (computer, operating system) to another?

## Precision and Accuracy

- How accurate must data calculations be?
- To what degree of precision must calculations be made?

## Time to Delivery / Cost

- Is there a prescribed timetable for development?
- Is there a limit on the amount of money to be spent on development or on hardware or software?

# Elaboration

- Analyze, model, specify...
- Some Analysis Techniques
  - Data Flow Diagrams (DFD)
  - Use case Diagram
  - Object Models (ER Diagrams)
  - State Diagrams
  - Sequence Diagrams
  - Activity Diagrams

# Requirements Modeling

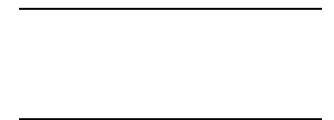
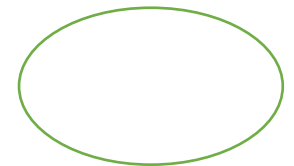
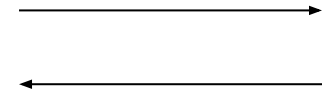
1. Scenario-based Models
  - User Stories
  - Use Case Diagram
2. Class-oriented Models
  - Class Diagram
  - CRC Cards
3. Behavioral Models
  - State Diagram
  - Sequence Diagram
4. Flow-oriented Models
  - Data Flow Diagram

# Data Flow Diagram(DFD)

- A Data Flow Diagram (DFD) is a traditional **visual representation of the information flows within a system**.
- By drawing a Data Flow Diagram, you can tell the information provided by and delivered to someone who takes part in system processes, the information needed to complete the processes and the information needed to be stored and accessed.
- The DFD is presented in a **hierarchical fashion**. That is, the first data flow model (sometimes called a **level 0 DFD** or **context diagram**) represents the system as a whole.
- Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

# Symbols used in DFD

- **Square Box:** A square box defines **external entities** (source or sink) of the system. Source supplies data to system and sink receives data from system.
- **Arrow or Line:** An arrow identifies the **data flow** i.e. it gives information to the data that is in motion. Connects processes, external entities and data stores.
- **Circle or bubble chart:** It represents as a **process** that gives us information. It is also called processing box.
- **Open Rectangle:** An open rectangle is a **data store** where data is stored. Data can be read from or written to the data store



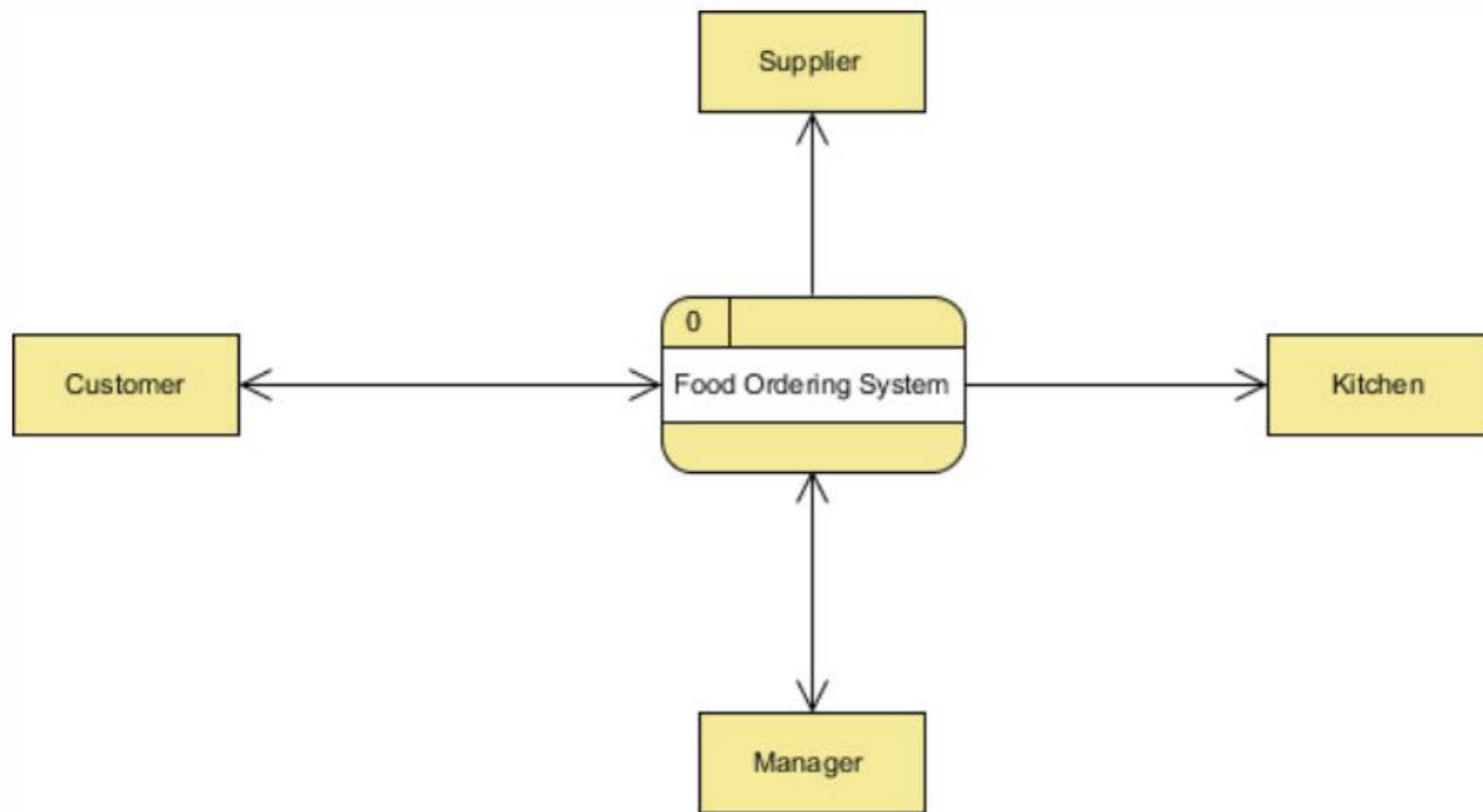
# Level-0 DFD

- A context diagram is a data flow diagram that only shows the **top level**, otherwise known as Level 0.
- At this level, there is only **one visible process node** that represents the functions of a complete system regarding how it interacts with external entities.
- Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

# Level-0 DFD(Food Ordering System)

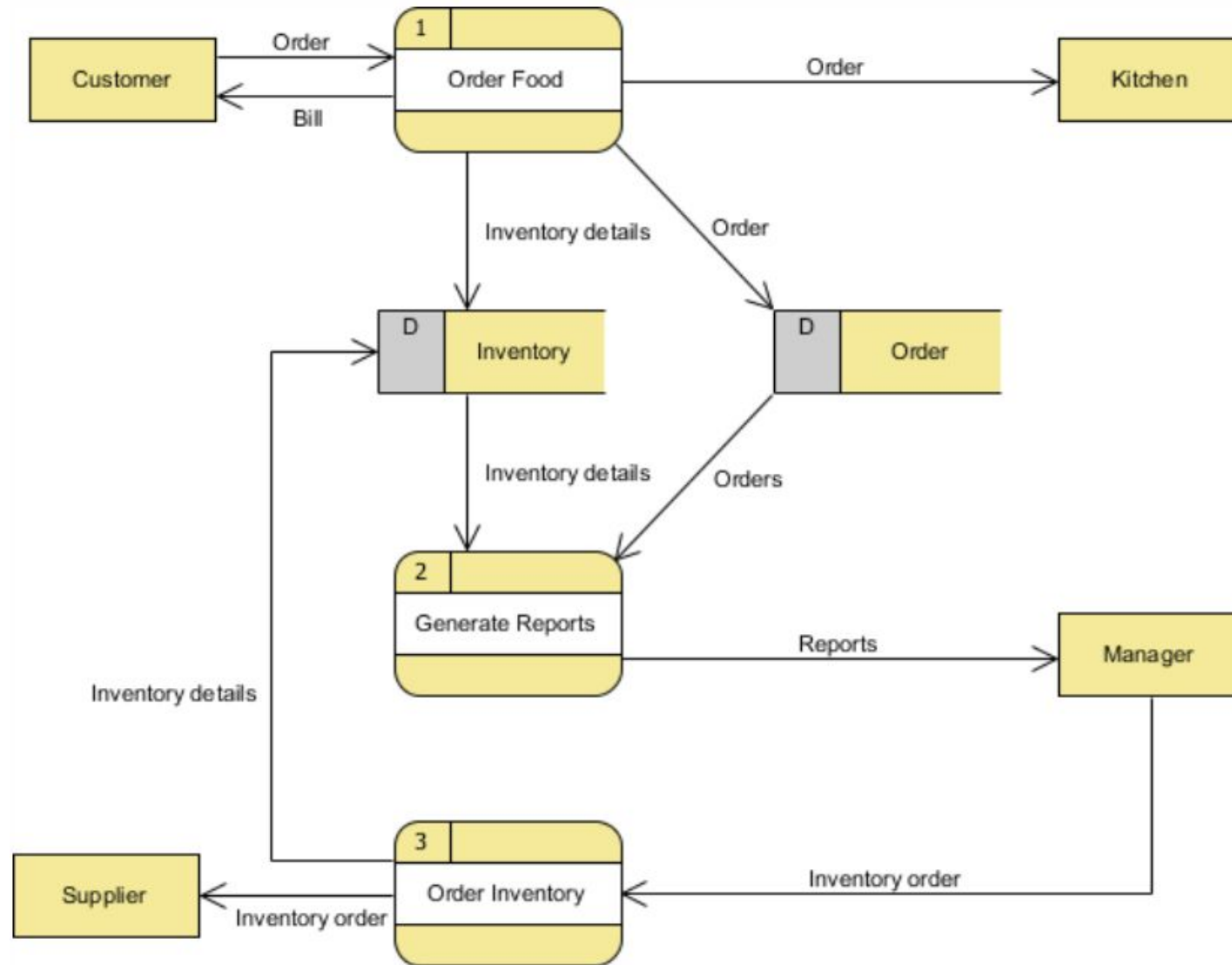
- It contains a process (shape) that represents the system to model, in this case, the "*Food Ordering System*".
- It also shows the participants who will interact with the system, called the external entities. In this example, the *Supplier*, *Kitchen*, *Manager*, and *Customer* are the entities who will interact with the system.
- In between the process and the external entities, there is data flow (connectors) that indicate the existence of information exchange between the entities and the system.

# Level-0 DFD(Food Ordering System)

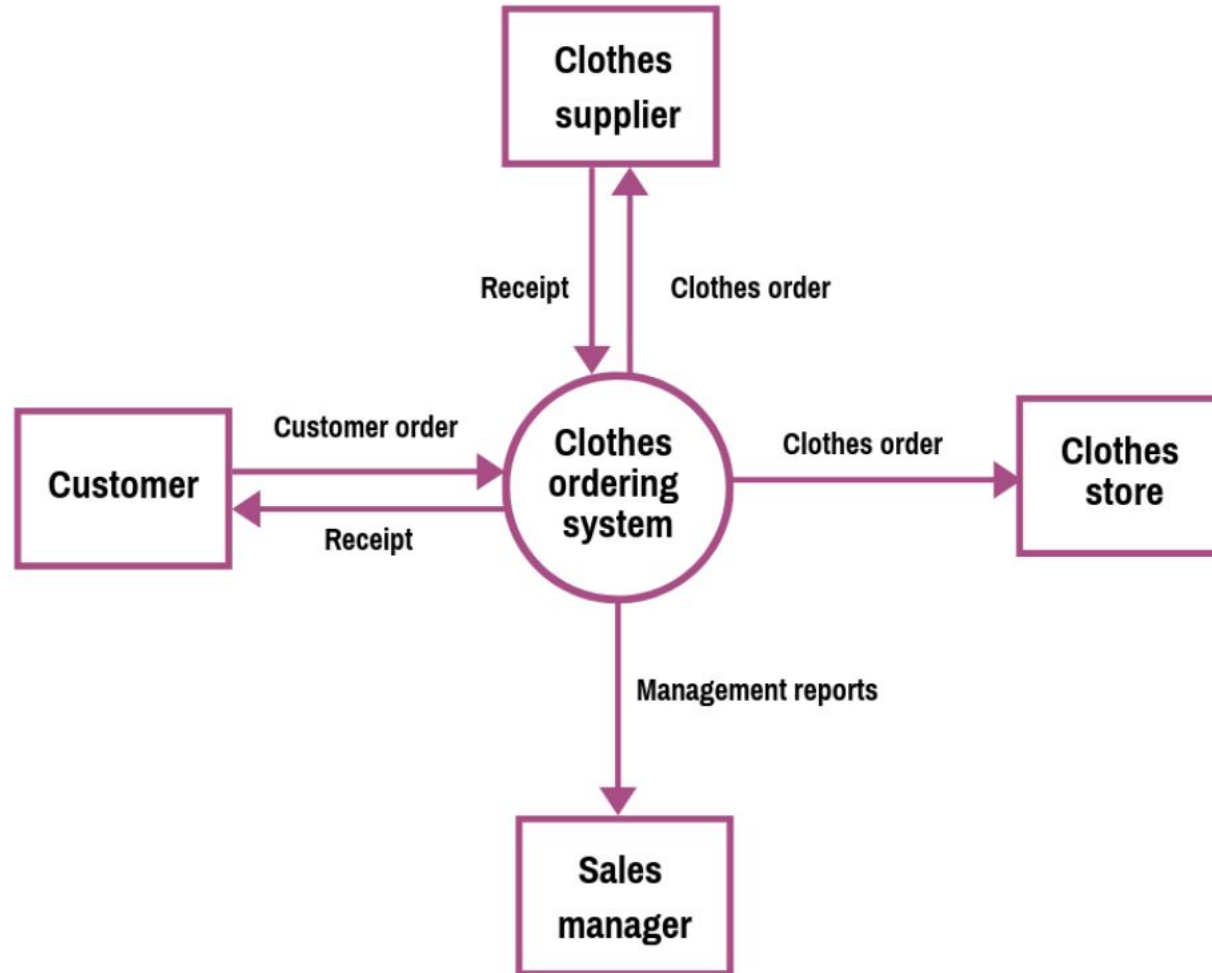




# Level-1 DFD (Food Ordering System)



# Level-0 DFD(Clothes Ordering System)



# Level-0 DFD(Clothes Ordering System)

**Step1:** Define the process.

As it is a context data flow diagram, the process is only one. In our case, it is *Clothes Ordering System*. Draw a rectangle for the process.

**Step 2:** Create the list of all external entities.

In our example, the external entities are: *Customer, Clothes Store, Clothes Supplier, and the Sales Manager*. These are all entities who are involved with our system. Also, now you can draw a rectangle for each of the entities.

**Step 3:** Create a list of the data flows.

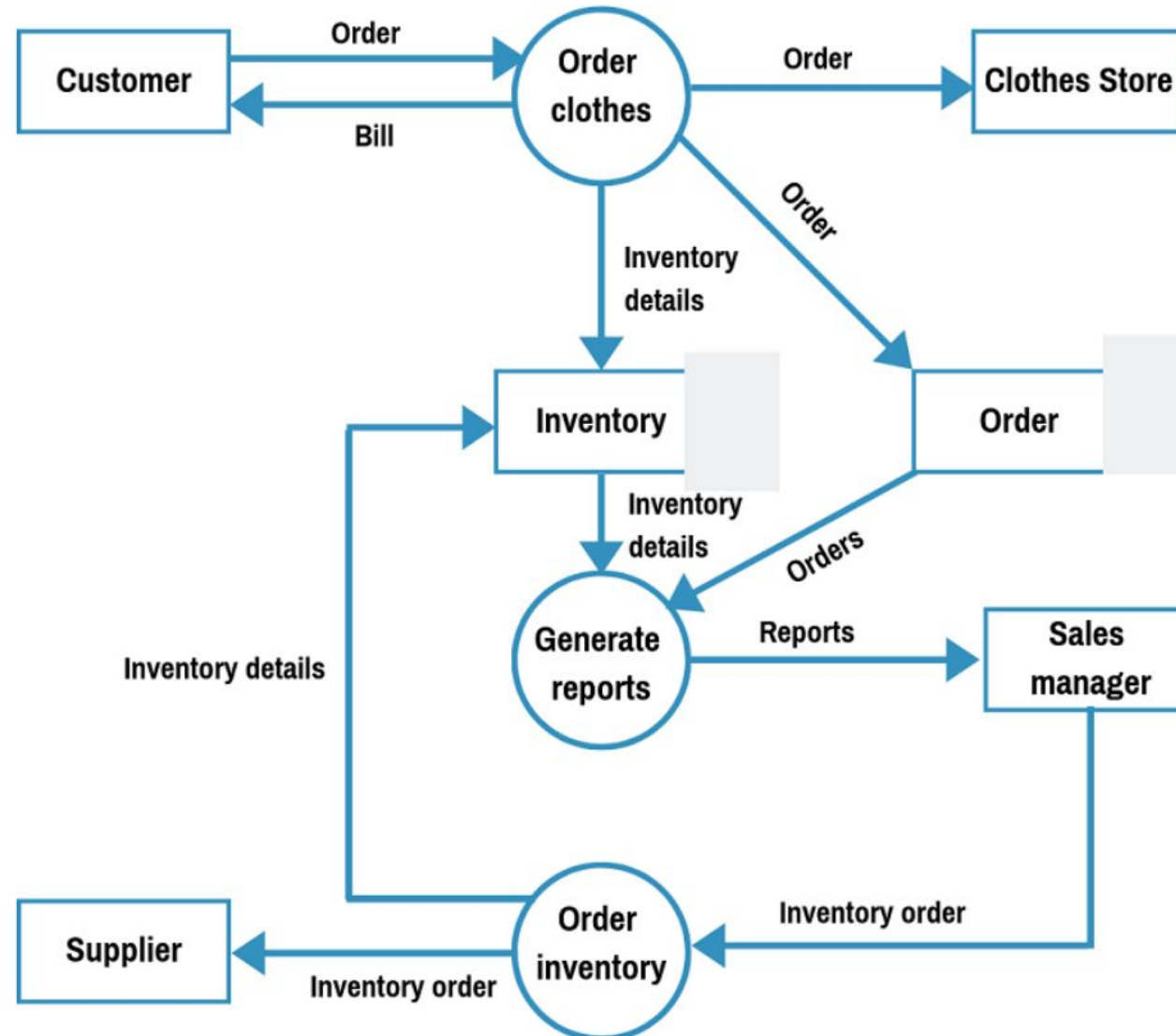
In between our process and the external entities, there are data flows that show a brief description of the type of information exchanged between the entities and the system.

In our example, the list of data flows includes: *Customer Order, Receipt, Clothes Order, Receipt, Clothes Order, and Management Report*.

Now, connect the rectangles with arrows signifying the data flows.

If data flows both ways between any two rectangles, create two individual arrows.

# Level-1 DFD(Clothes Ordering System)



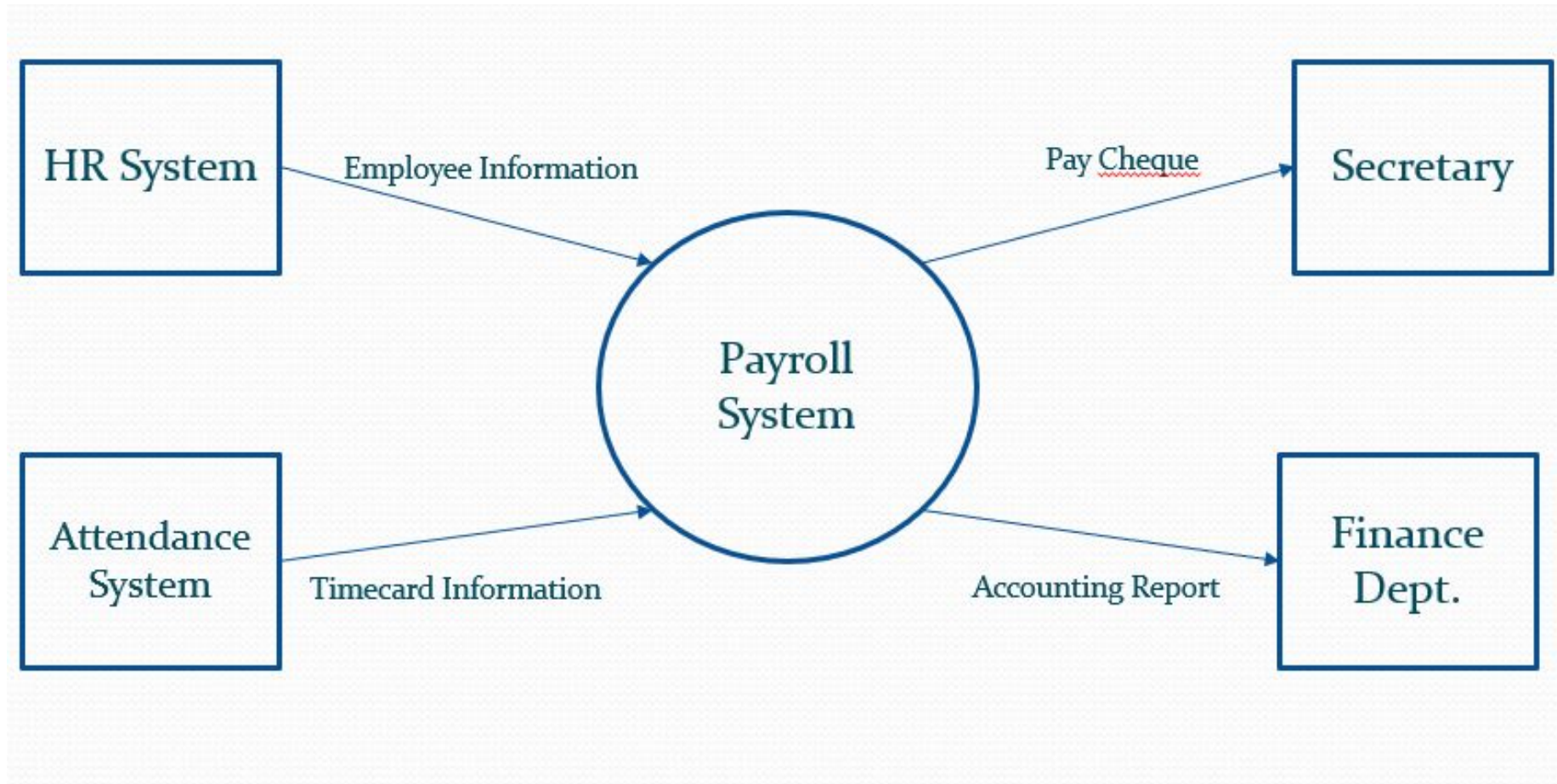
# Level-1 DFD(Clothes Ordering System)

- Level-0 DFD contains only one process and does not illustrate any data store. This is the main difference with level 1 DFD.
- Level 1 DFD breaks down the main process into subprocesses that can then be seen on a deeper level. Also, level 1 DFD contains data stores that are used by the main process.

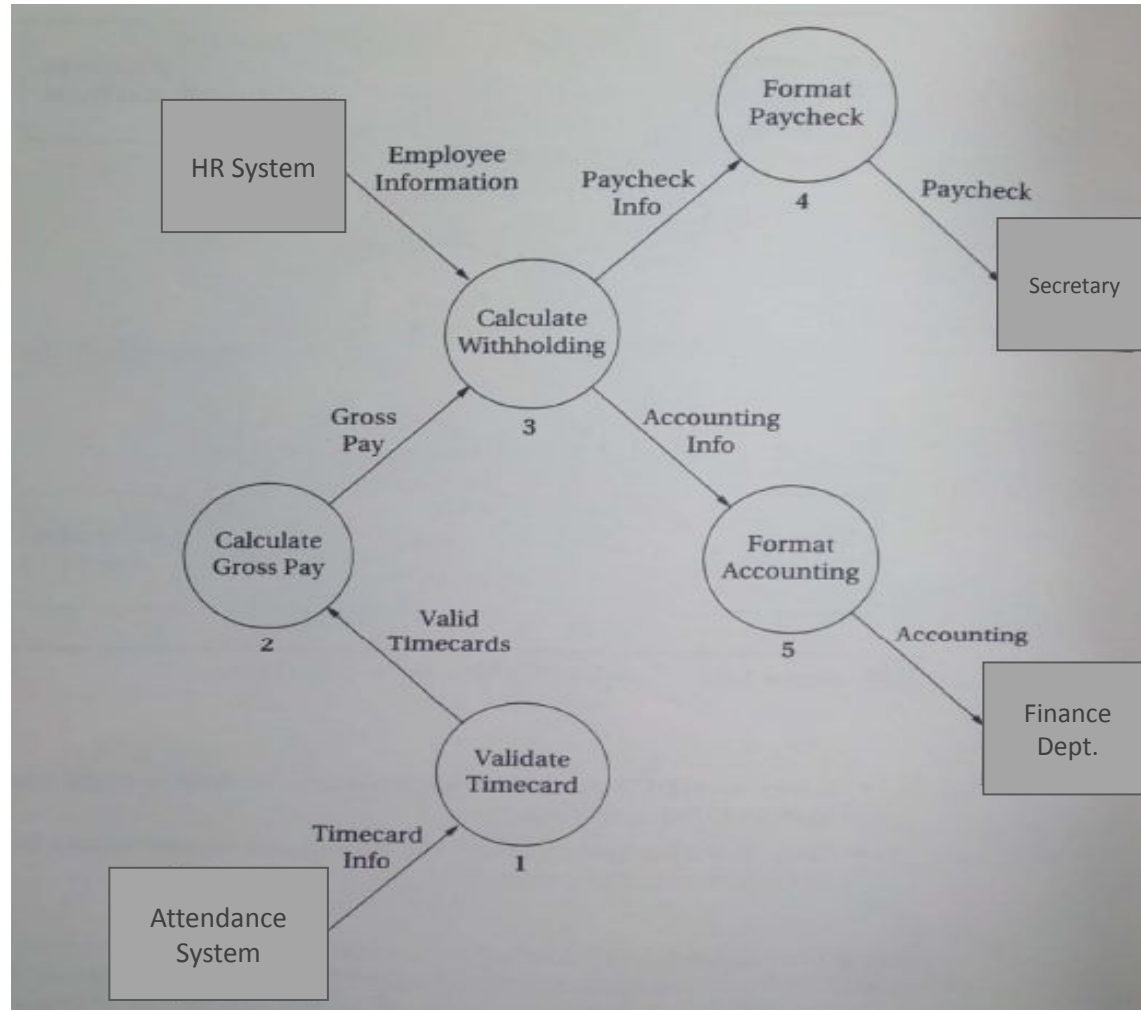
# Level-1 DFD(Clothes Ordering System)

- **Step 1:** Define the processes.
- The three processes are: *Order Clothes, Generate Reports, and Order Inventory.*
- **Step 2:** Create the list of all external entities.
- The external entities are: *Customer, Clothes Store, Sales Manager, and Supplier*
- **Step 3:** Create the list of the data stores.
- These are: *Order and Inventory*
- **Step 4:** Create the list of the data flows
- Data flows are: *Order, Bill, Order, Order, Inventory details, Inventory details, Orders, Reports, Inventory Order, Inventory Order, Inventory details.*
- **Step 5:** Create the diagram.

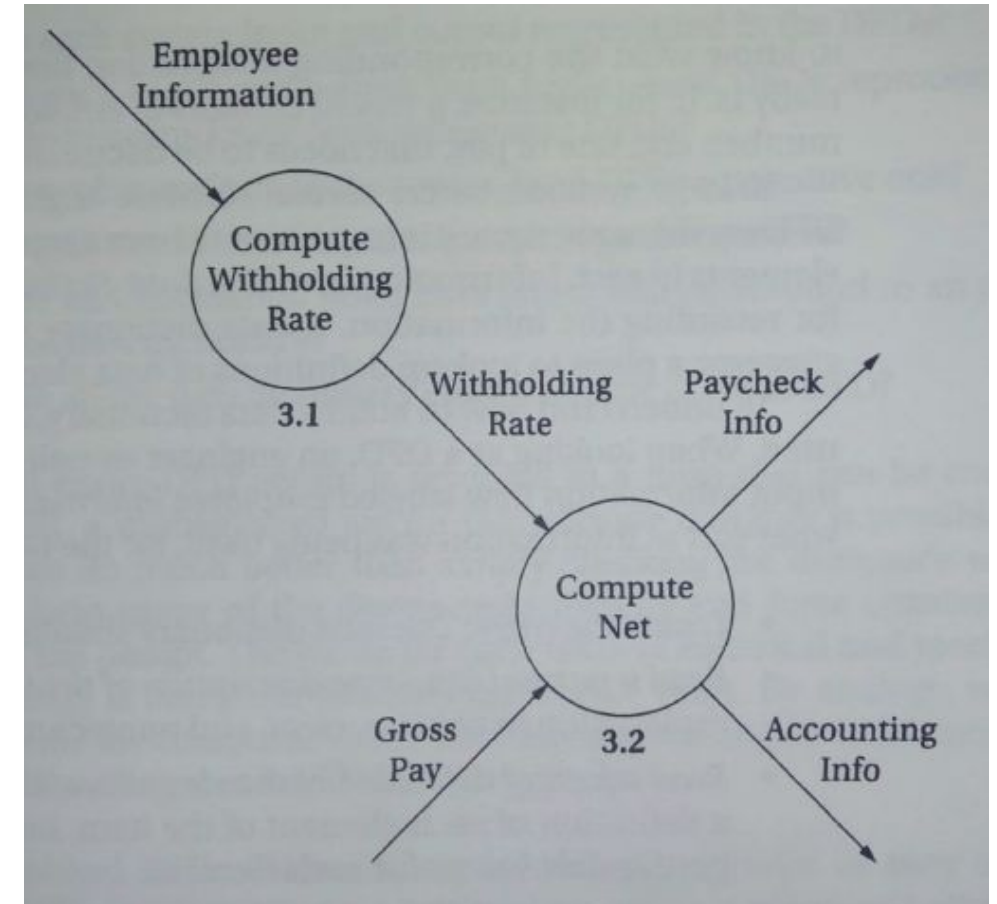
# Level-0 DFD(Payroll System)



# Level-1, Level-2 DFD(Payroll System)



Level-1



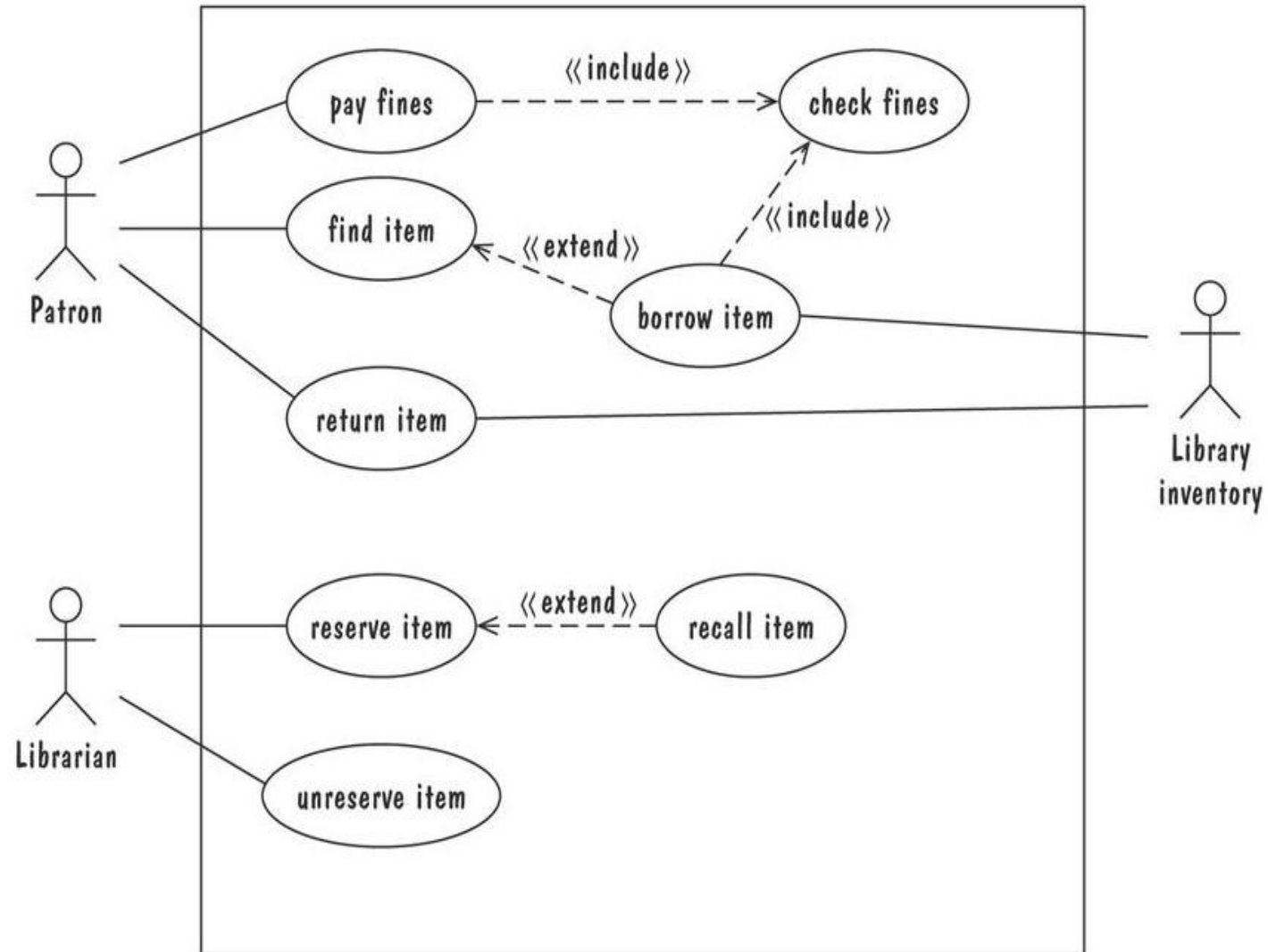
Level-2



# Use Case Diagram

- Components
  - **A large box:** *system boundary*
  - **Stick figures** outside the box: *actors*, both human and systems
  - Each **oval** inside the box: a use case that represents some major required functionality and its variant
  - A **line** between an actor and use case: the actor participates in the use case
- Use cases do not necessarily model all the tasks, instead they are used to specify user views of essential system behaviour

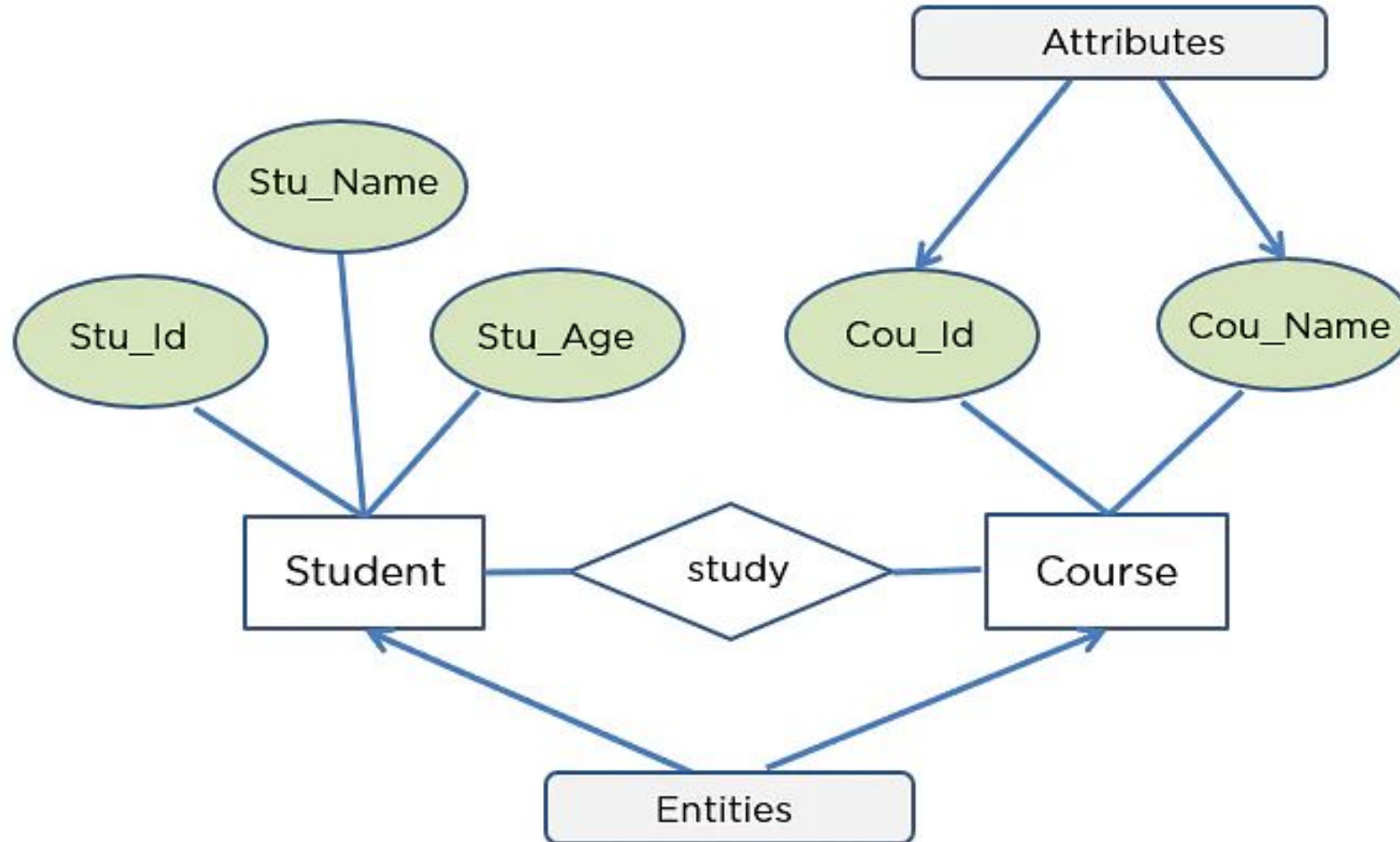
# Use Case Diagram



# Entity Relationship Diagram

- ER Model is used to model the **logical view** of the system from a data perspective.
- It consists of following components:
  - **An *entity***: depicted as a **rectangle**, represents a collection of real-world objects that have common properties and behaviours.
  - **A *relationship***: depicted as an **edge** between two entities, **with diamond** in the middle of the edge specifying the type of relationship.
  - **An *attribute***: represented as an **oval**, describes data or properties associated with the entity.

# Entity Relationship Diagram



# Entity Relationship Diagram

- ER diagrams are popular because
  - they provide an **overview of the problem** to be addressed.
  - the **view is relatively stable** when changes are made to the problem's requirements.
- ER diagram is likely to be used to model a problem early in the requirements process.

# Class Diagram

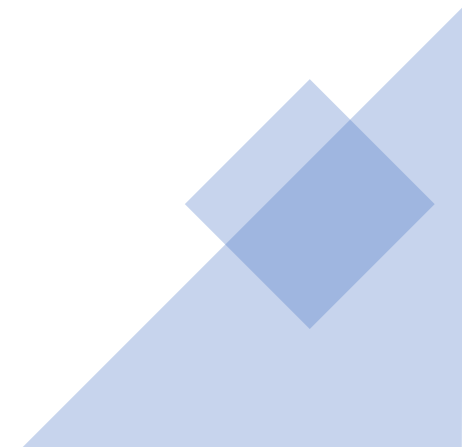
- Class Diagram is a type of **static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- **Components:**
  - ***Objects***: akin to entities, organized in classes.
  - ***Attributes***: object's variables or characteristics.
  - ***Behaviours***: actions on the object's variables.

# Types of Relationship

1. **Association:** Represents static relationship between classes.
2. **Dependency:** Represents dependency of one class on another class.
3. **Aggregation:** Represents has-a relationship. Contained class does not have strong dependency on container class.
4. **Composition:** Contained class has a strong dependency on container class.
5. **Generalization:** Represents is-a relationship. It has superclass and subclass relationships

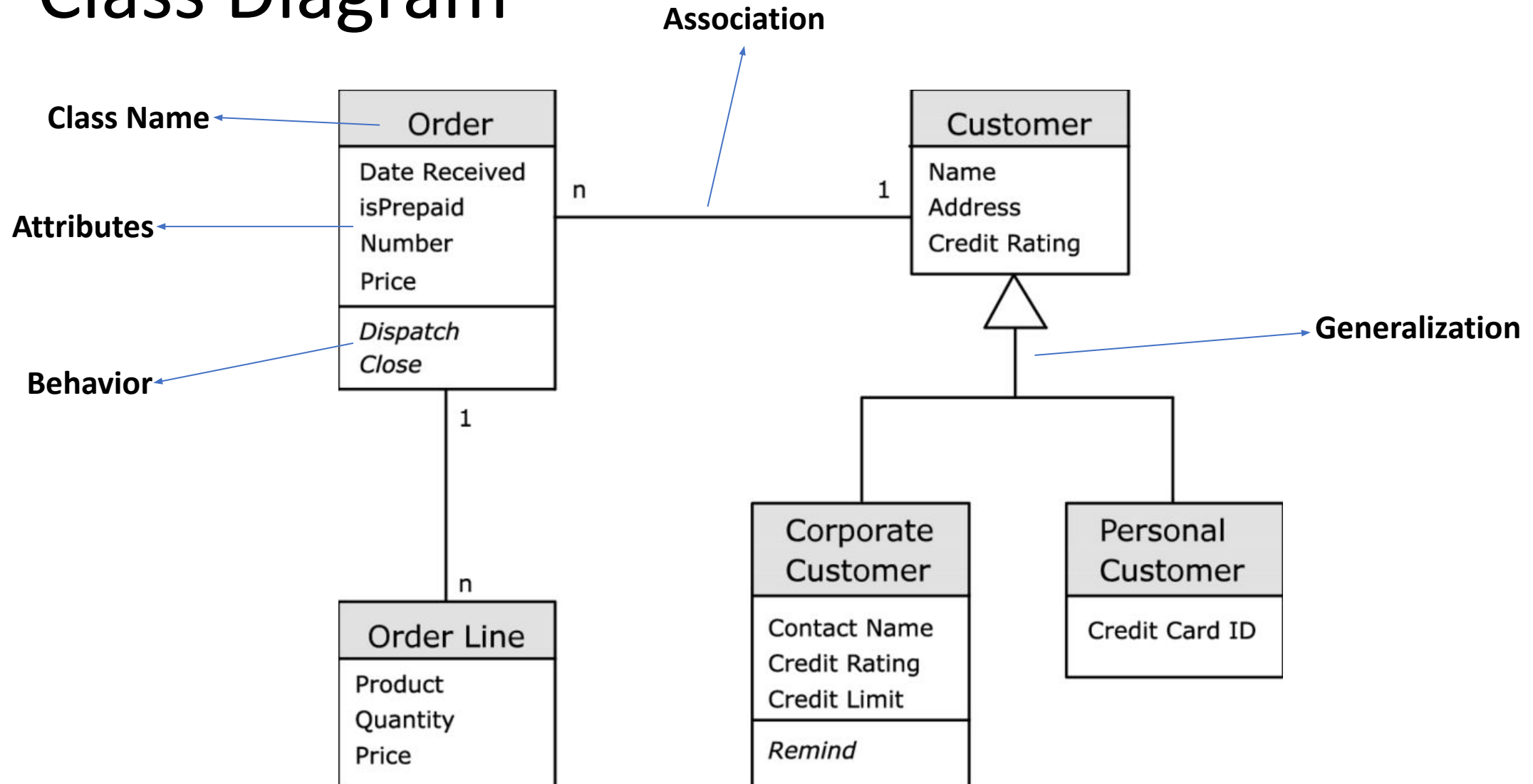


# Identifying Classes

- External Entities (producers, consumers)
  - Things
  - Occurrences or events
  - Roles
  - Organizational Units
  - Places
  - Structures
- 



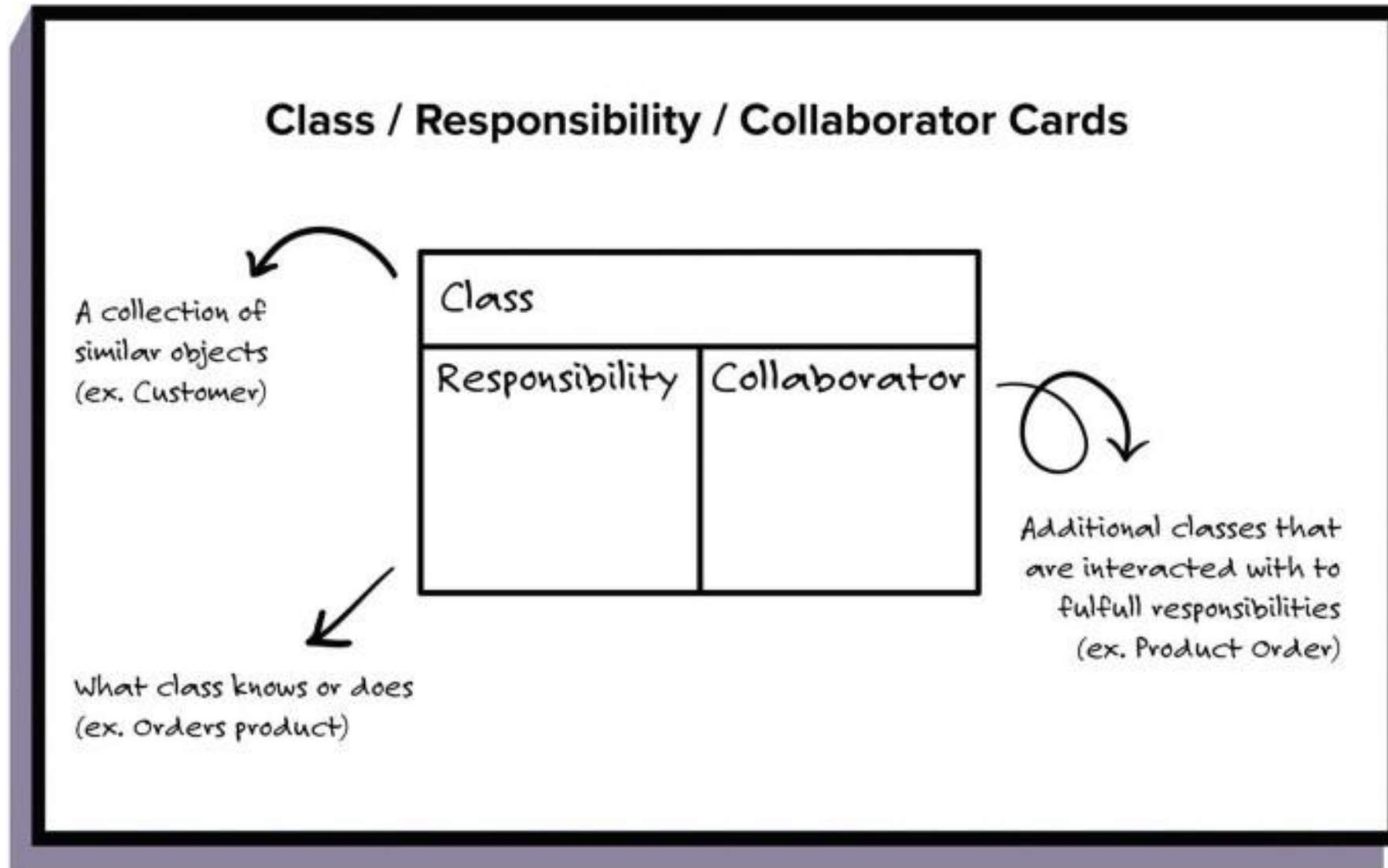
# Class Diagram



# Class-Responsibility-Collaborator Modeling

- Class-responsibility-collaborator (CRC) modeling provides a simple means for **identifying and organizing the classes** that are relevant to system or product requirements.
- A CRC model can be viewed as a **collection of index cards**.
- Each index card contains a list of **responsibilities** on the left side and the corresponding **collaborations** that enable the responsibilities to be fulfilled on the right side.
- **Responsibilities** are the attributes and operations that are relevant for the class.
- **Collaborators** are those classes that provide a class with the information needed or action required to complete a responsibility.

# Class-Responsibility-Collaborator Modeling



# Class-Responsibility-Collaborator Modeling

Class Sales	
Responsibility	Collaboration
<ul style="list-style-type: none"><li>• Knowledge</li><li>• Behaviour</li><li>• Operation</li><li>• Promotion</li><li>...</li></ul>	<ul style="list-style-type: none"><li>• Partner</li><li>• Clients</li><li>...</li></ul>

Class Transaction	
Responsibility	Collaboration
<ul style="list-style-type: none"><li>• Money Transfer</li><li>• Auditing</li><li>...</li></ul>	<ul style="list-style-type: none"><li>• Card reader</li><li>• Clients</li><li>...</li></ul>

Class Order	
Responsibility	Collaboration
<ul style="list-style-type: none"><li>• Price</li><li>• Stock</li><li>• Valid Payment</li><li>...</li></ul>	<ul style="list-style-type: none"><li>• Customers</li><li>• Order line</li><li>...</li></ul>

Class Delivery	
Responsibility	Collaboration
<ul style="list-style-type: none"><li>• Item identity</li><li>• Check Receiver</li><li>• Order No.</li><li>• Total Qty</li><li>...</li></ul>	<ul style="list-style-type: none"><li>• Partner</li><li>• Clients</li><li>...</li></ul>

# State Diagram

- A graphical description of all dialog between the system and its environment.
  - **Node** (*state*) represents a stable set of conditions that exists between event occurrences
  - **Edge** (*transition*) represents a change in behaviour or condition due to the occurrence of an event
- Useful both for **specifying dynamic behaviour** and for **describing how behaviour should change** in response to the history of events that have already occurred.

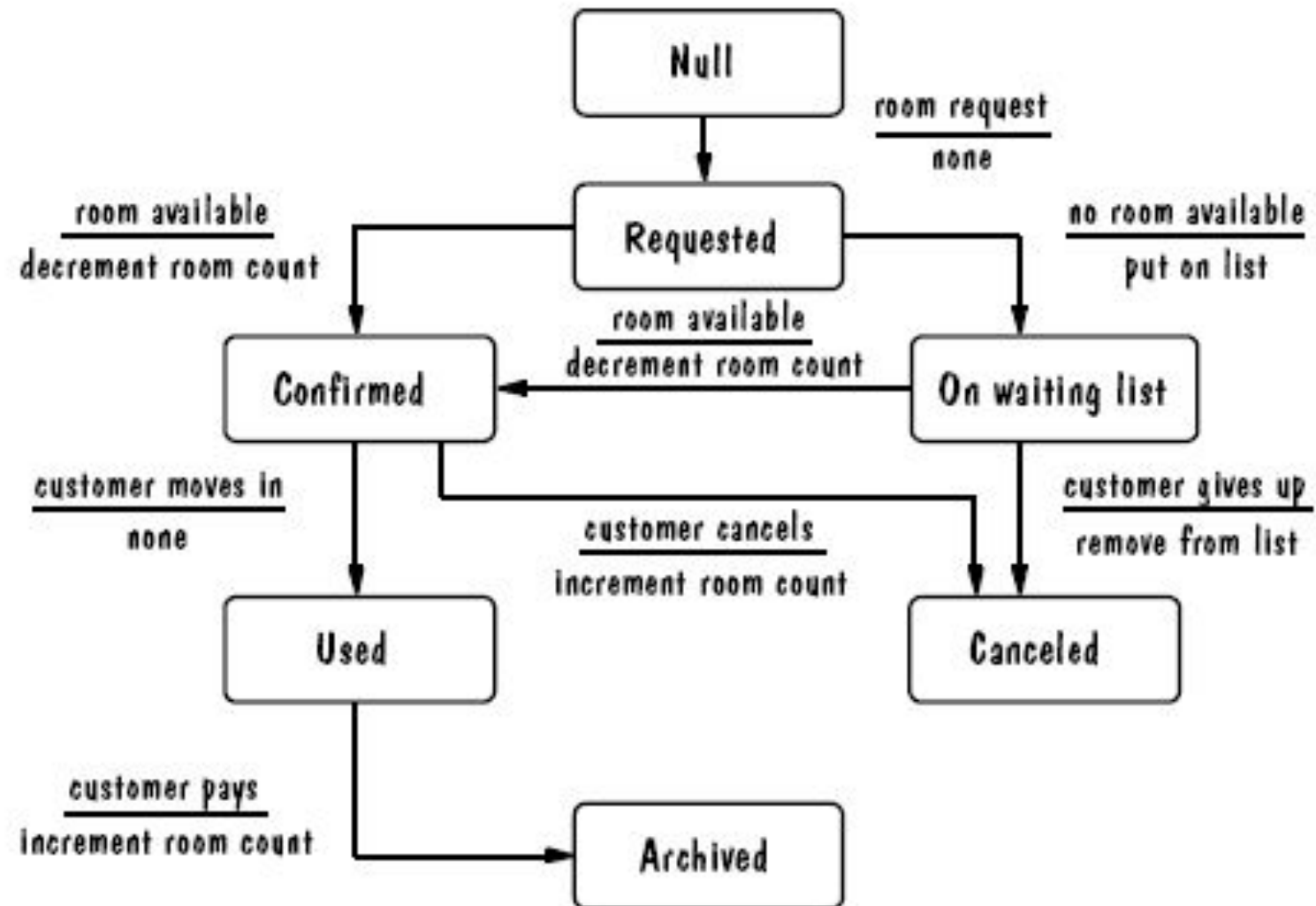
# State Diagram: UML Notation



# State Diagram: UML Statechart

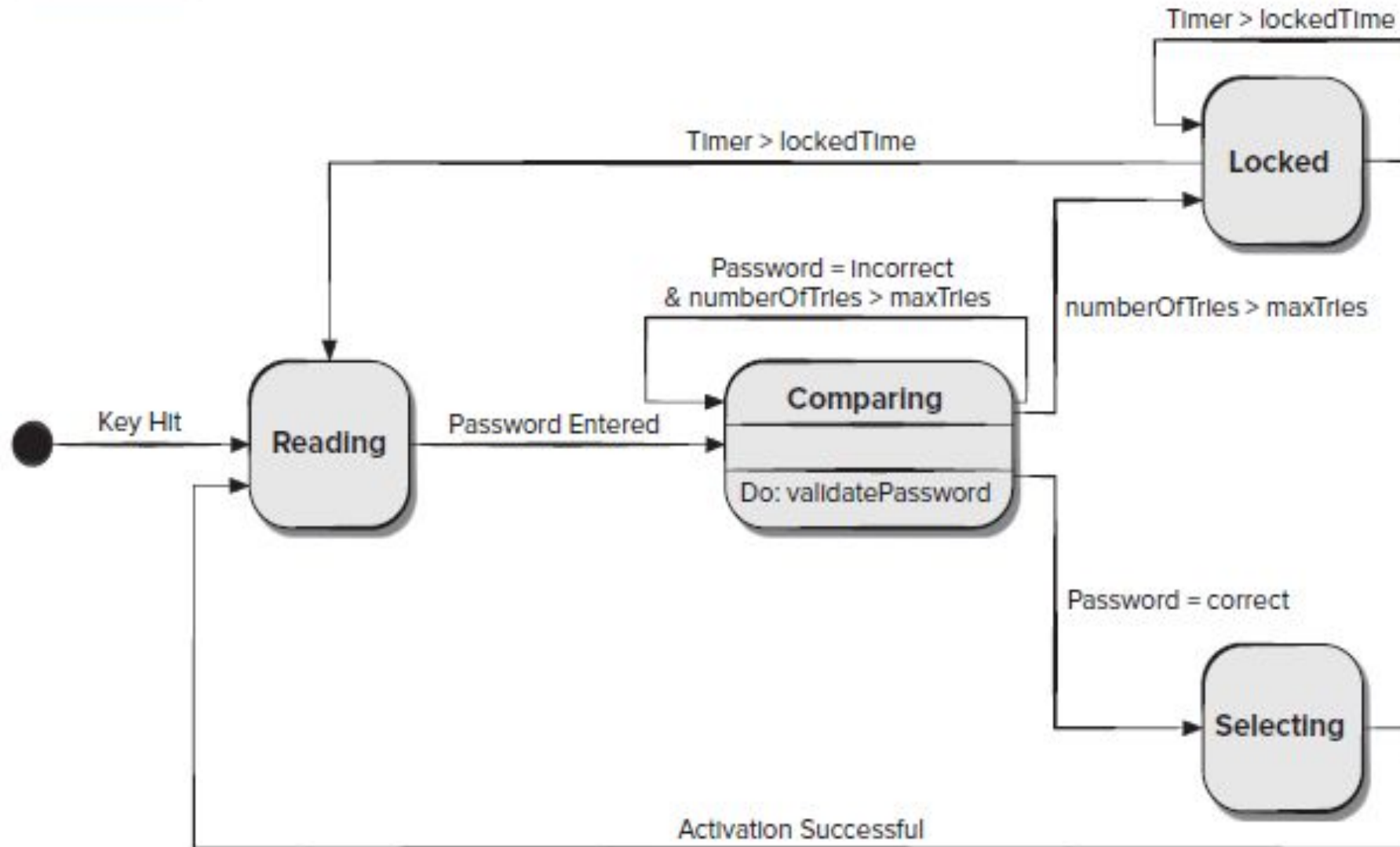
- Statechart diagrams provide us an efficient way to **model the interactions or communication** that occur within the external entities and a system.
- These diagrams are used to **model the event-based system**.
- A UML model is a **collection of concurrently executing statecharts**.
- UML statechart diagram have a rich syntax, including **state hierarchy, concurrency, and inter-machine communication**.

# State Diagram





# State Diagram(Control Panel)

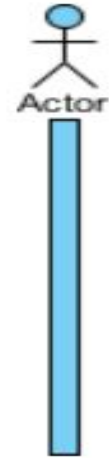


# Sequence Diagram

- A sequence diagram simply **depicts interaction between objects in a sequential order** i.e., the order in which these interactions take place.
- Sequence diagrams describe **how and in what order the objects in a system function**.

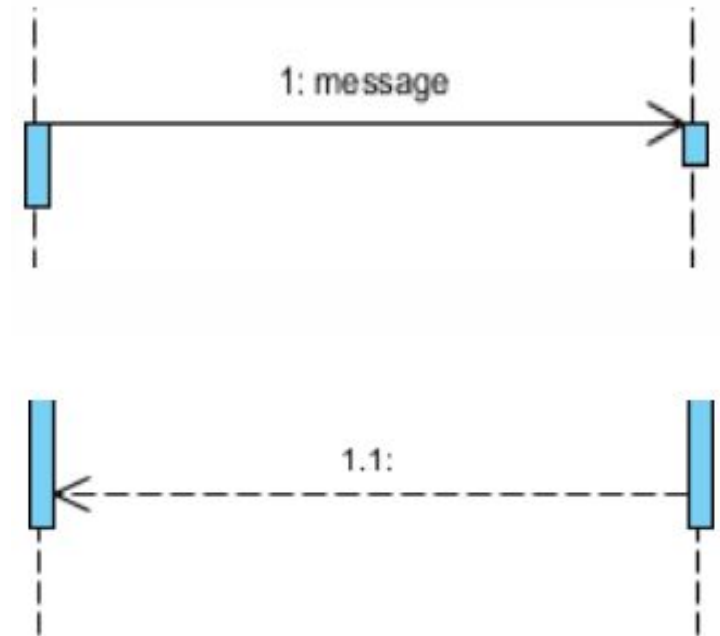
# Sequence Diagram Notations

- Actor:
  - Represent roles played by human users, external hardware, or other subjects.
- Lifeline:
  - A lifeline represents an individual participant in the interaction.



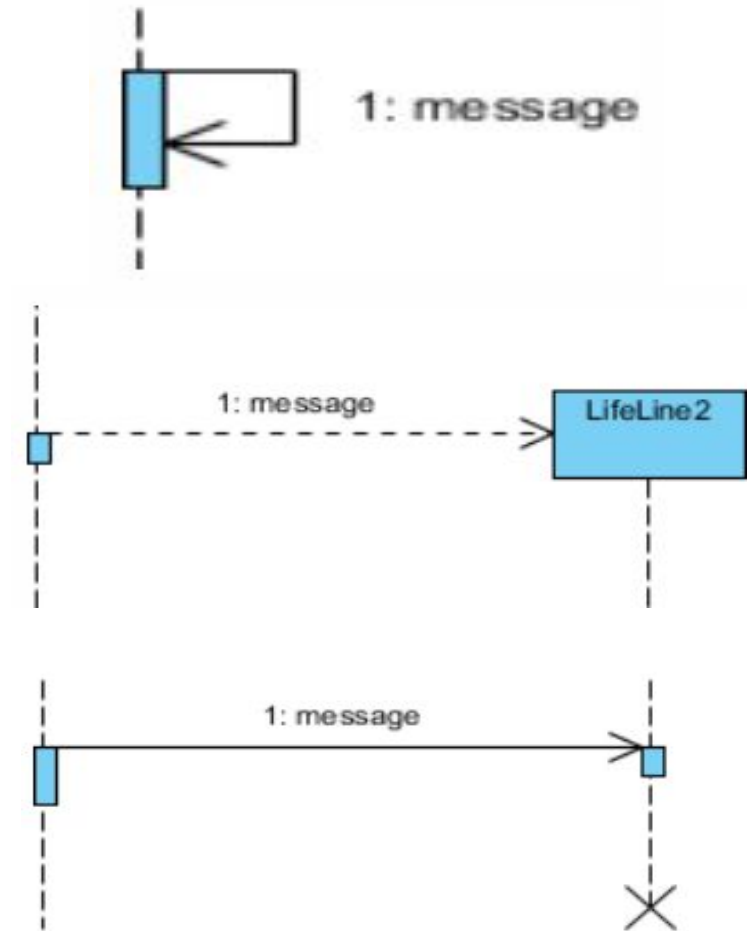
# Sequence Diagram Notations

- Call Message:
  - It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.
- Return Message:
  - It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



# Sequence Diagram Notations

- Self Message:
  - It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.
- Create Message:
  - It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.
- Destroy Message:
  - It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



## Search Book : Use Case

### - Main scenario -

The Customer specifies an author on the Search Page and then presses the Search button.

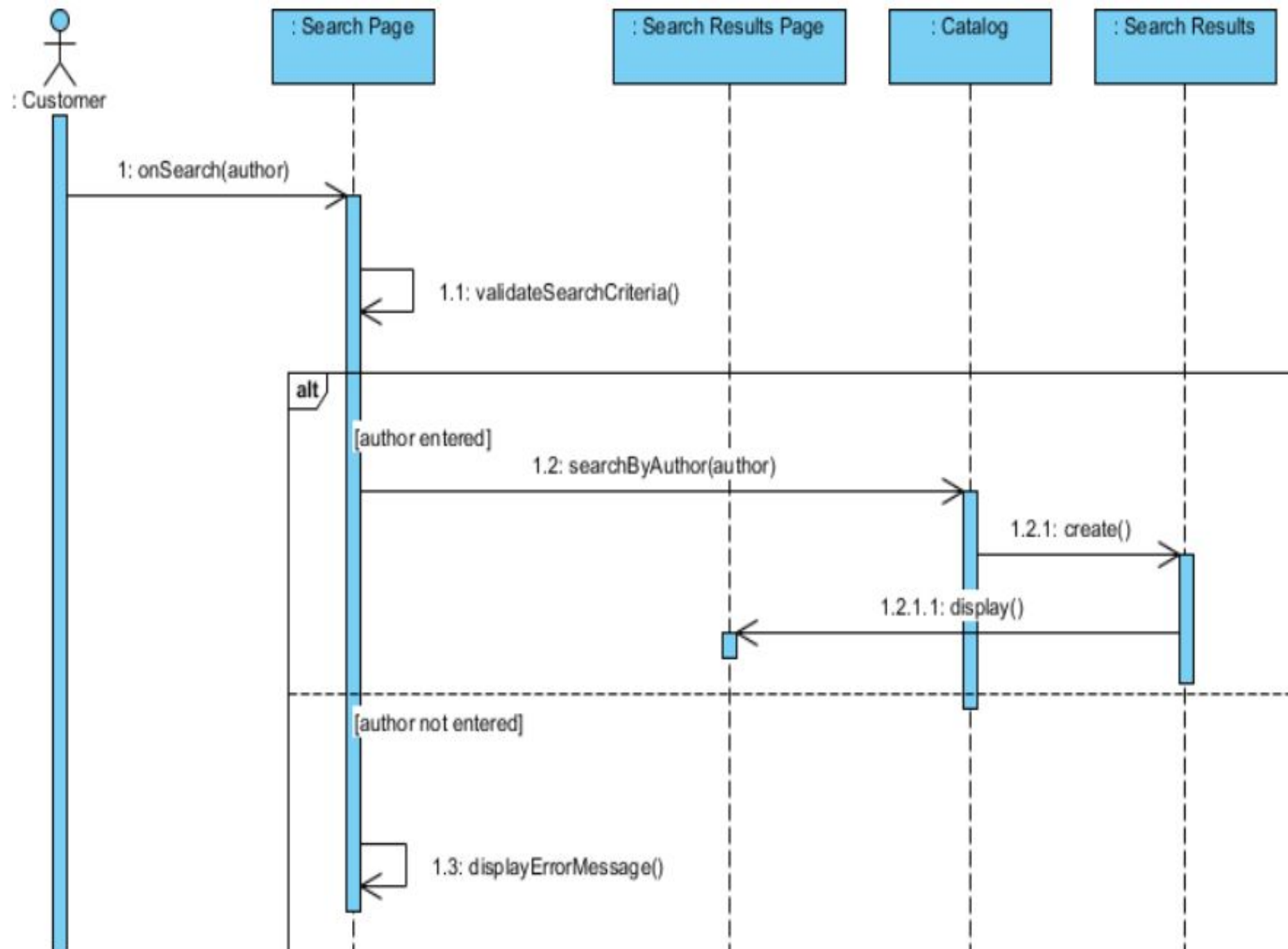
The system validates the Customer's search criteria.

If author is entered, the System searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results page.

### - Alternate path -

If the Customer did not enter the name of an author before pressing the Search button, the System displays an error message



# Swimlane Diagrams

- A swimlane diagram is a type of flowchart that **defines who does what in a process.**
- Using the **metaphor of lanes in a pool**, a swimlane diagram provides clarity and accountability by placing process steps within the horizontal or vertical “swimlanes” of a particular employee, work group or department.
- It shows connections, communication and handoffs between these lanes, and it can serve to **highlight waste, redundancy and inefficiency in a process.**

## SOFTWARE DEVELOPMENT

## SOFTWARE DESIGN

## SOFTWARE CODING

