SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chennai-603203.

FACULTY OF ENGINEERING AND TECHNOLOGY

School of Computing

Department of Data Science and Business Systems

Academic Year (2022–2023)

18CSC268J - Software Design with UML

SEMESTER–IV

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chennai-603203.

FACULTY OF ENGINEERING AND TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this is the Bonafide record of work done by **M.S.V.KARTHIK** **RA2111042010058** of IV semester B.Tech COMPUTER SCIENCE WITH BUSSINESS SYSTEMS during the academic year 2022-2023 in the

18CSC268J - Software Design with UML.

S.JEEVA
Staff -Incharge

Dr. M. Lakshmi
Head of the Department

Submitted for the practical examination held on at SRM Institute of Science and Technology, Kattankulathur, Chennai-603203

Examiner-1

Examiner-2

SESSION	DATE	EXERCISE	SIGN
1	19-01-23	5 OPEN-SOURCE TOOLS FOR UML	
2	25-01-23	UML PACKAGE DIAGRAM	
3	03-02-23	UML PACKAGE DIAGRAM WITH DRILL DOWN FEATURES	
4	10-02-23	CASE STUDY ON STATE-CHART AND ACTIVITY DIAGRAM	
5	22-02-23	UML STATE-CHART DIAGRAM	
6	22-02-23	UML ACTIVITY DIAGRAM	
7	09-03-23	UML USE CASE AND CLASS DIAGRAM	
8	16-03-23	UML COMPONENT DIAGRAM	
9	24-03-23	UML INTERACTION DIAGRAM	
10	10-04-23	UML COLLABORATION DIAGRAM	
11	18-04-23	UML DEPLOYMENT DIAGRAM	
12	25-04-23	UML THREADS	

EXPERIMENT-1 OPEN-SOURCE TOOLS FOR UML DIAGRAMS

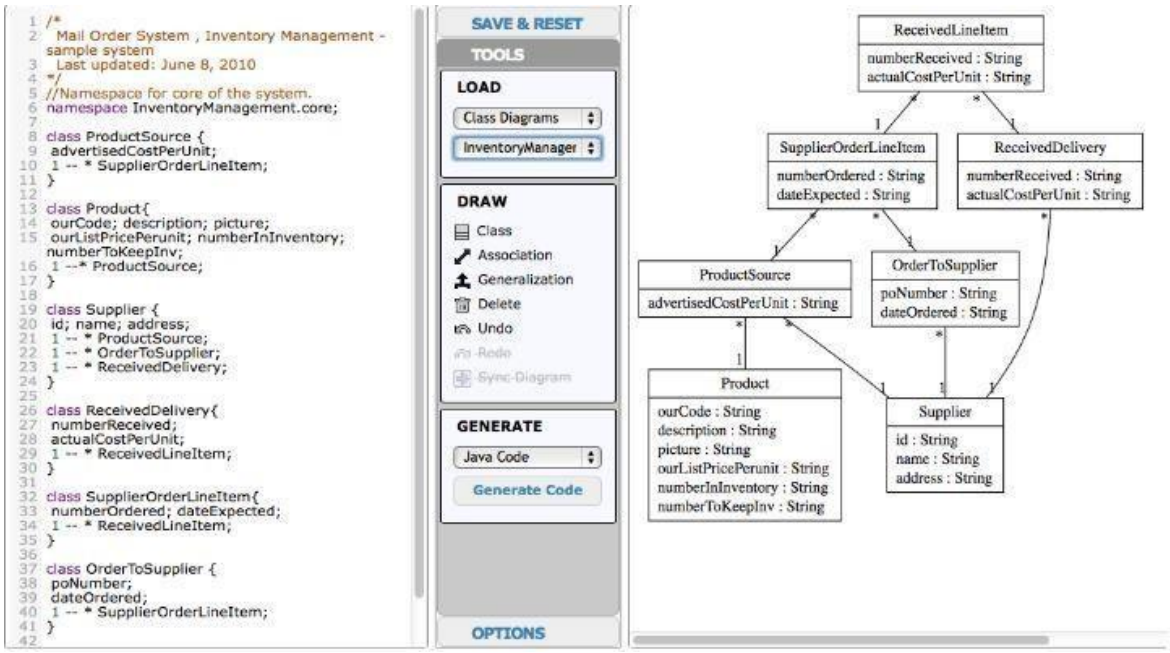
AIM: Understanding and studying about various open source tools to draw UML diagrams.

UMPLE

- Umple is an open-source modelling tool for software developers and students to make UML in the fastest way in their classroom.
- Its works online as an eclipse plugin and as a stand-alone command-line Jar.
- Umple is a model-oriented programming technology that adds UML associations and state machines to Java and PHP.
- It is used to draw UML diagrams, embeds models in code, and generate a complete system.

Features:

- Simple and saves time
- It makes you model in UML textually.
- You can generate top quality code from class diagrams and state diagrams
- You can add a little bit of Umple code into an existing Java, PHP, or Ruby system
- Umple works like a pre-processor



Limitations:

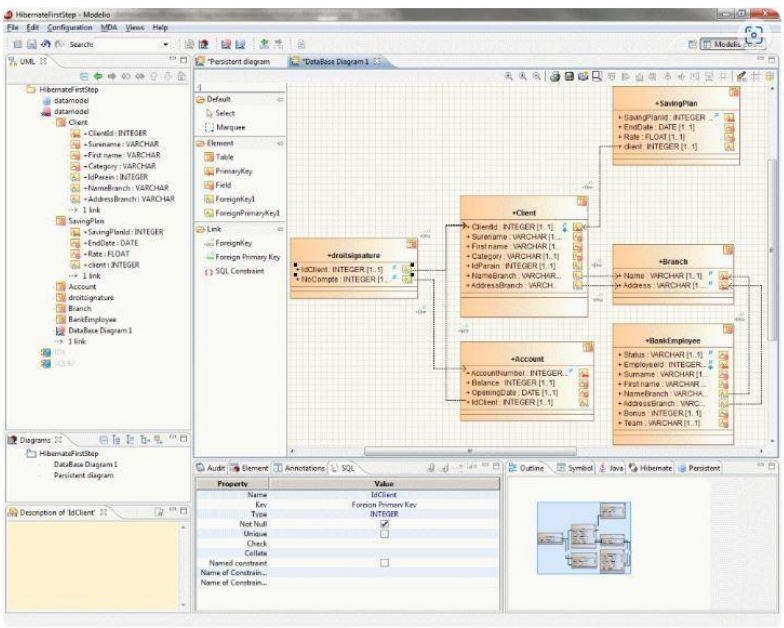
- Making it possible to develop software simultaneously with a textual and visual representation of high-level abstractions.
- Reducing the volume of code that needs to be written due to code from abstractions.
- Umple is explicitly not a single new programming language instead, it consists of a set of
- features that extend multiple existing programming languages.

MODELIO

- Modelio is the first modeling environment.
- The tool combines BPMN support and UML support.
- It is one of the best free UML tools that provide support for a wide range of models and diagrams.
- It is accessible to the user as an open-source UML diagram tool for developing UML diagrams.

Features:

- Modelio offers an XMI import/export feature that enables you to exchange UML2 models between various tools.
- It is one of the best open-source UML tool that offers you to export diagrams in SVG, Gif or JPEG format.
- You can extend Modelio for any language, methodology, or modeling technique.
- It is a free and open-source HTML5 online flowchart software
- It provides multiple diagram options cloud storage
- Supports PNG, JPEG, WMF, XML, GIF and BMP file formats
- It Provides import/export options for XML, XMI, EMF, PNG, JPEG and BMP
- Support programming language like C++, C#, Java and SQL
- Offers Multi-file support, Drag and Drop, and Print option



Limitations

- BPMN documentation is not exported nor imported
- Builds fail because of missing directories in build properties
- Unable to look for installed plugins
  Observation of translucent defects

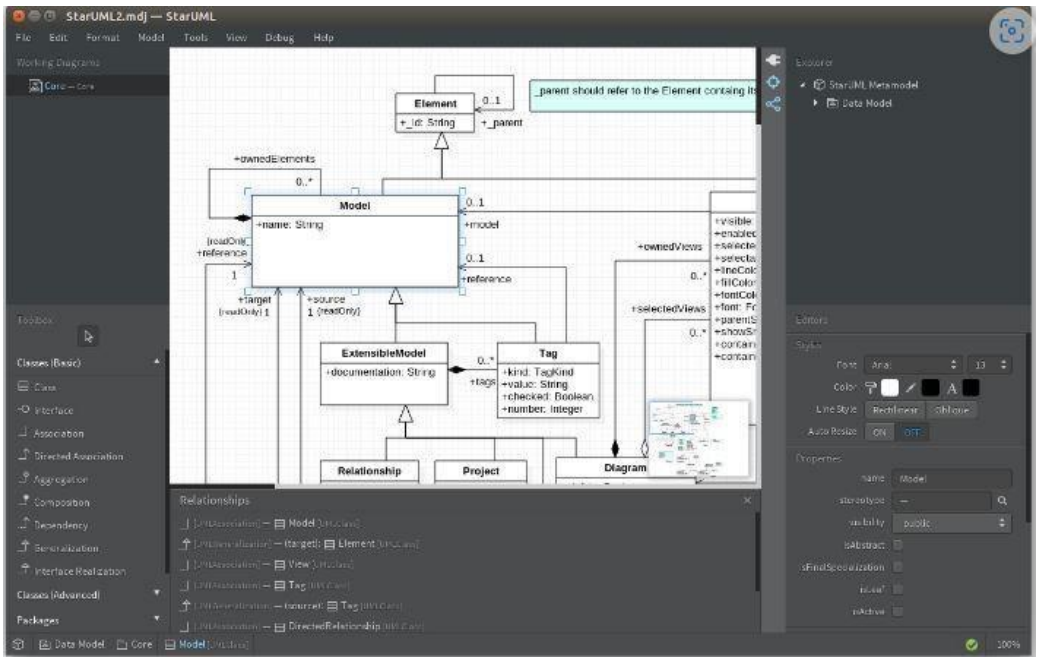
-

STAR UML

- StarUML is a software modeler for agile and concepts modeling for macOS windows and different types of Linux(Ubuntu, Debian, Redhat, fedora).
- It is one of the best UML software that provides eleven types of diagram.
- it is support code generation for various programming languages such as Java, C#, C++, and python, it also has Open APIs

Features:

- This UML diagram software allows you to discover and install third-party extensions.
- No limit to using this commercial software for evaluation.
- Offers pre-built templates for Class Diagram
- It provides multiple diagram options cloud storage
- Supports JSON, PNG, HTML and JPEG file formats
- It Provides import/export options for PDF, XML, PNG, JPEG, HTML and SVG
- Support programming language like C, C++, C#, Java and Python
- Provides Custom Uml Profile, Auto Update, Model-driven Development, Fast Modeling, Dark And Light Themes and Code Markdown Support
- Offers Multi-file support, Drag and Drop, and Print option



Limitations

- Ascending Degree of Complexity
- Architecture – Indifferent Design
- Support for new languages in case of reverse engineering
  Classes with template parameters do not export class as a template class

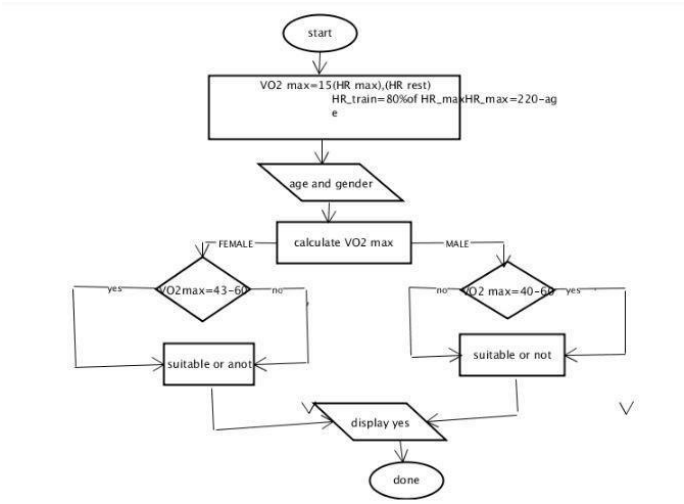
-

DRAW.IO

- Draw.IO is a free online UML tool.
- It is one of the best UML tools that allows users to create and manage the drawing easily these tools.
- A lot of the wide and early share is available with this tool.

Features:

- No limit on the number of sizes
- Templates are present in the software design itself.
- This free UML diagram tool allows you to save the model in your preferred location
- Offers pre-built templates for diagram
- Seamlessly integrates with Aha, Atom, Bioiocons, BookStack, Docstell, FOSWiki, Grafana, Growi, JupyterLab, Lark, LumApps, Nextcloud, Nuclino, Redmine and Sandstorm
- It provides multiple diagram options cloud storage
- Supports PNG, JPEG, SVG, JSON, XML, CSV and PDF file formats
- It Provides import/export options for PNG, JPEG, SVG, JSON, XML, CSV and PDF
- Offers firewall based encryption
- Provides Keep your diagram data secure, Diagram wherever you want, Collaborate in real-time with shared cursors, Easy-to-use diagram editor, and Many advanced tools
- Offers Version history, Drag and Drop, and Print option



. Limitations

- As you work in a browser, the app seems to lag if you work on it for a while
- The app appears to glitch from time to time if you work for longer periods of time,.
- Making you lose some of your work of you’re not careful.
- Exporting your work in the form you would like can be challenging.

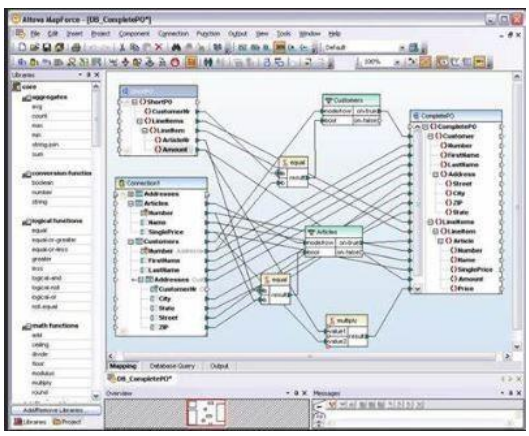
-

VISUAL PARADIGM

- Visual Paradigm is a software design tool that is tailored for engine software projects.
- This UML editor tool helps the software development team to model business information systems and development processes.

Features:

- It offers a complete tool like for process analysis, system design, database design, etc.
- Offers user story feature to capture and maintain users' needs.
- Offers pre-built templates for Social Media, Events, Business, Marketing, Documents, School, Personal, Infographics, Posters and Gift Card
- Seamlessly integrates with Eclipse, NetBeans, IntelliJ IDEA, Visual Studio and Android Studio
- Free plan offers 1GB cloud storage
- It provides multiple diagram options 1GB cloud storage
- Supports JPG, PNG, SVG, TIFF, EMF, PDF and DOC file formats
- It Provides import/export options for JPG, PNG, SVG, TIFF, EMF, PDF and XML
- Offers 256-bit SSL encryption
- Provides Visual Modeling, Enterprise Architecture, Business Analysis & Design, Project Management, Agile & Scrum Development, Online Diagram Tool, Spreadsheet Tool, and Team Collaboration
- Offers Multi-file support, Drag and Drop, and Print option



Limitations

- Ascending Degree of Complexity
- Architecture – Indifferent Design
- Support for new languages in case of reverse engineering Classes with template parameters do not export class as a template class

RESULT: Hence the study of different open-source to draw UML diagrams has been studied successfully.

EXPERIMENT-2 UML PACKAGE DIAGRAM

AIM: To draw a UML package diagram for an order system of application using Star UML.

DESCRIPTION:

A package diagram is a type of Unified Modelling Language (UML) diagram that shows the organization and structure of related components in a software system. The purpose of a package diagram is to provide a high-level view of the system's components and how they are related to one another.

In a package diagram, the components of the system are grouped into packages, which are represented as rectangles with a tab. The name of the package is written inside the rectangle, and the packages can be nested within one another to indicate their relationships. Packages can contain classes, interfaces, and other packages, and the relationships between packages can be shown using directed arrows.

A package diagram can also show dependencies between packages, which indicate that one package relies on another to function properly. These dependencies are represented by directed arrows pointing from the dependent package to the package it depends on.

PURPOSE:

Package diagrams are useful for modelling the structure and organization of a system, and they provide a high-level view that can be easily understood by stakeholders and other team members. They are also useful for identifying potential problems in the design early on and making changes before they become more complex and costly to resolve

HOW TO DRAW:

Identify the system's components: Analyse the requirements of the system and identify the major components or functionalities that need to be modelled.

Determine the packages: Group related components into packages, where each package represents a functional unit of the system.

Create class diagrams for each package: Create class diagrams for each package, including classes, their attributes, and relationships between classes.

Represent packages and dependencies: Use UML notation to represent the packages and their dependencies on one another. The package symbol is a rectangle with a tab, and dependencies between packages are represented by directed arrows pointing from the dependent package to the package it depends on.

Add optional components: If necessary, add additional components, such as interfaces, to the packages to further refine the design.

Review and refine the diagram: Review the diagram to ensure that it accurately represents the system and its components, and make any necessary revisions.

Document the diagram: Document the diagram with meaningful names and descriptions to help others understand the design

Package diagram for food delivery service

A package diagram for a food delivery service might include the following packages and classes:

Order Package:

- Order class: to manage customer orders, including details such as items ordered, delivery address, payment information, etc.
- Menu class: to store information about food items available for ordering, such as name, description, price, etc.

Delivery Package:

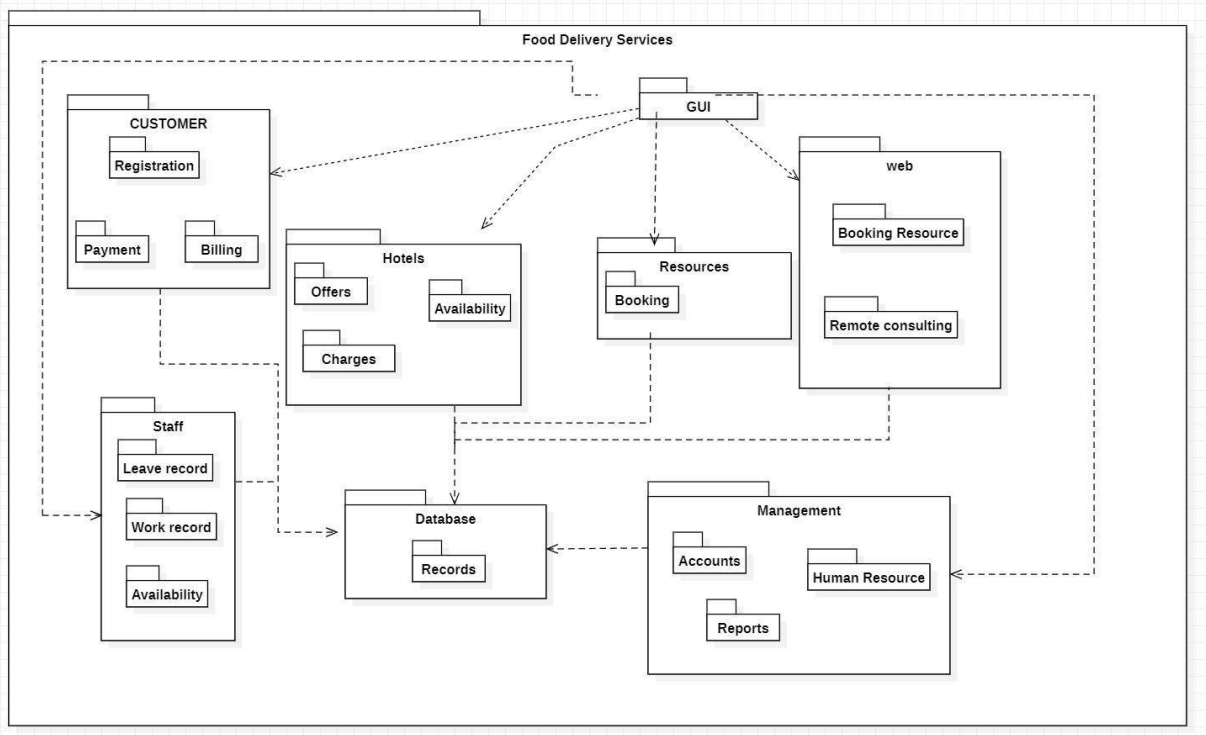
- Delivery class: to manage the delivery process, including details such as delivery status, delivery address, delivery driver, etc.
- Driver class: to store information about delivery drivers, including name, contact information, availability, etc.

Payment Package:

- Payment class: to manage payment processing, including details such as payment method, payment status, etc.
- Credit Card class: to store information about a customer's credit card, including name, number, expiration date, etc.

Customer Package:

- Customer class: to store information about customers, including name, contact information, order history, etc.
- These packages and classes can be connected with various relationships, such as inheritance, aggregation, and association, to model the relationships and interactions between different parts of the system.



PACKAGE DIAGRAM FOR FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML package diagram for an order system of application.

EXPERIMENT-3 UML PACKAGE DIAGRAM WITH DRILL DOWN FEATURES

AIM: To draw a UML package diagram with drill down features for an order system of application using Star UML.

DESCRIPTION:

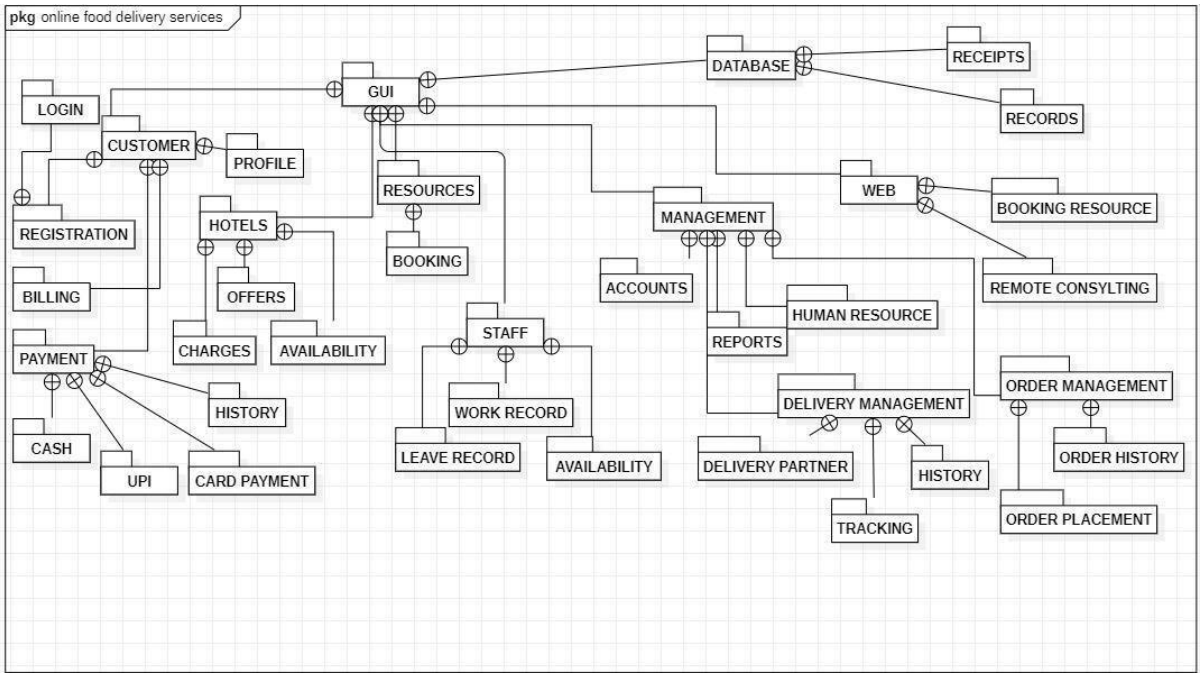
In UML, drill down feature allows the user to navigate from a higher-level view of a system to a lower-level view of the system's components. It provides a way to explore the details of a system by breaking down a complex diagram into smaller and more detailed diagrams. This is particularly useful when dealing with complex systems or large diagrams, as it allows users to focus on specific parts of the diagram while ignoring the rest. By drilling down into a diagram, users can gain a deeper understanding of the system and its components.

PURPOSE:

The purpose of drill down features is to allow users to navigate from a high-level view of data or information to a more detailed and specific view. It enables users to explore and analyse data at different levels of granularity and depth. By providing the ability to drill down into the details, users can gain insights into specific aspects of the data and identify patterns or trends that may not be immediately apparent at a higher level. This feature can be useful in various applications such as business intelligence, data analytics, and information systems.

HOW TO DRAW:

- Start by drawing the main package that you want to represent in your diagram.
- Identify the sub-packages or modules that make up the main package.
- Draw each sub-package as a smaller package inside the main package, using a nesting or containment relationship between them.
- Add any relevant classes or interfaces to the sub-packages, and draw these inside the sub-package boxes.
- Identify any drill down features that you want to include in your diagram. These could be methods, attributes, or other elements of the classes or interfaces.
- Draw these drill down features as additional boxes inside the sub-packages or classes, with a dependency or association relationship to the main package or class.
- Use appropriate labels and annotations to make the diagram clear and easy to understand.
- Repeat the process for any additional levels of drill down that you want to include in your diagram, nesting sub-packages or classes inside each other as needed.



DRILL DOWN FOR A PACKAGE DIAGRAM FOR ONLINE FOOD SERVICES

RESULT: Thus, we have successfully drawn the UML package diagram with drill down features for an order system of application.

EXPERIMENT-4 STATE AND ACTIVITY DIAGRAM

AIM: To study about state and activity diagrams.

State diagrams and activity diagrams are two types of UML diagrams used to model the behaviour of systems. They have different purposes and structures, but they share some common components:

Components of a State Diagram:

State: A state represents a condition or situation of an object or system. It is usually represented as a rectangle with rounded corners and labeled with the name of the state.

Transition: A transition represents a change of state in response to an event or a condition. It is usually represented as an arrow connecting two states, with the event or condition triggering the transition labeled on the arrow.

Initial state: An initial state represents the starting point of a state machine. It is usually represented as a filled circle connected to the initial state by a transition.

Final state: A final state represents the end point of a state machine. It is usually represented as a filled circle with no outgoing transitions.

Components of an Activity Diagram:

Activity: An activity represents a task or operation performed by an object or system. It is usually represented as a rectangle with rounded corners and labelled with the name of the activity.

Control flow: Control flow represents the order of execution of activities. It is usually represented as arrows connecting activities, with a diamond-shaped decision node to represent branching and merging of flows.

Initial node: An initial node represents the starting point of an activity diagram. It is usually represented as a filled circle connected to the initial activity by a control flow.

Final node: A final node represents the end point of an activity diagram. It is usually represented as a filled circle with no outgoing control flow.

Fork and Join nodes: Fork and join nodes are used to represent parallel execution of activities. A fork node splits the control flow into multiple paths, while a join node merges multiple paths into a single path. They are represented as solid lines with two or more branches or merging paths.

Comparison of state diagram and activity diagram

State diagrams and activity diagrams are two types of UML diagrams used to model the behaviour of systems, but they have different purposes and structures. Here are some of the key differences between state diagrams and activity diagrams:

Purpose: State diagrams are used to model the behaviour of objects or systems that have a finite number of states and transitions between them, while activity diagrams are used to model the flow of activities or processes within a system.

Elements: State diagrams include states, transitions, events, and actions, while activity diagrams include activities, control flow, decisions, and forks/joins.

Structure: State diagrams are composed of states and transitions, which describe the different states of the system and how it transitions between them in response to events. Activity diagrams are composed of activities and control flow, which describe the flow of activities or processes within a system.

Focus: State diagrams focus on the internal behaviour of an object or system, while activity diagrams focus on the interactions between objects or systems.

Representation of parallelism: State diagrams do not have a built-in mechanism for representing parallel execution, while activity diagrams have forks and joins to represent parallelism.

Termination: State diagrams terminate when the system reaches a final state, while activity diagrams terminate when all activities have been completed.

Conclusion:

state diagrams are used to model the behaviour of objects or systems with a finite number of states and transitions between them, while activity diagrams are used to model the flow of activities or processes within a system. State diagrams focus on internal behaviour, while activity diagrams focus on interactions between objects or systems, and activity diagrams have a built-in mechanism for representing parallel execution.

RESULT: Hence, the study of state and activity diagrams has been done successfully.

EXPERIMENT-5 UML STATECHART DIAGRAM

AIM: To draw a UML state chart diagram for an order system of application using Star UML.

DESCRIPTION:

A state chart diagram is a type of UML (Unified Modeling Language) diagram used to represent the behavior of a system or object. It shows the various states of an object or system and the transitions between these states in response to events or conditions.

PURPOSE:

The purpose of a state chart diagram is to model the dynamic behavior of a system or application, by showing the different states that an object or entity can be in, and the events that cause it to transition from one state to another. Some of the key purposes of a state chart diagram include:

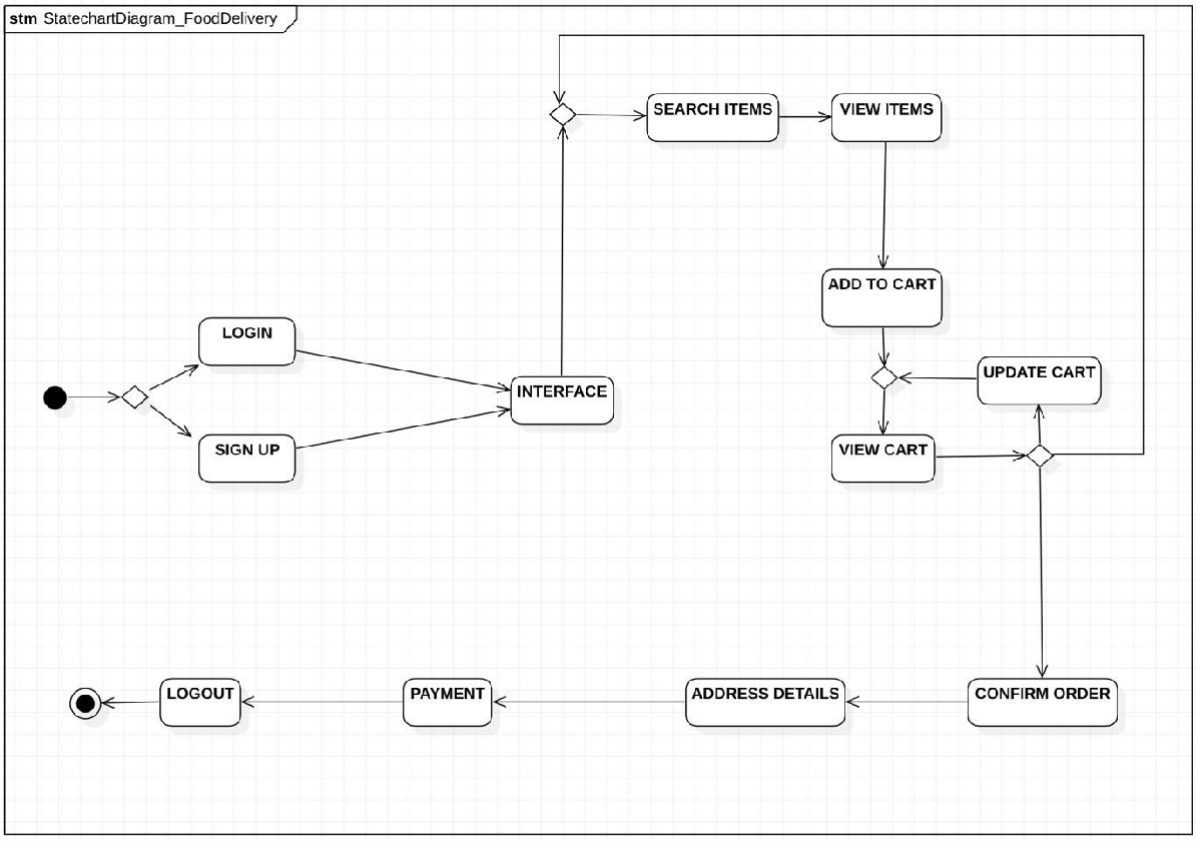
- Modeling system behavior
- Visualizing system behavior
- Communication tool
- Testing tool

HOW TO DRAW:

- Identify the object or entity: The first step is to identify the object or entity that you want to model with the state chart diagram. This could be a system, a device, or an entity such as a user or a customer.
- Identify the states: Once you have identified the object or entity, the next step is to identify the different states that it can be in. States represent different conditions or modes that the object or entity can be in, such as “idle,” “processing,” or “complete.”
- Identify the events: The next step is to identify the events that cause the object or entity to transition from one state to another. Events represent external stimuli or inputs that cause the object or entity to change its state, such as a user clicking a button or a system receiving a message.
- Define the transitions: Once you have identified the states and events, the next step is to define the transitions between the states. Transitions represent the movement from one state to another, and are triggered by events. Transitions can have conditions or actions associated with them, which describe what happens when the transition occurs.
- Draw the diagram: The final step is to draw the state chart diagram, using UML notation to represent the states, events, and transitions. The state chart diagram should be easy to read and understand, and should accurately represent the behavior of the object or entity being modeled.

The main components of a state chart diagram are:

- States: These are the different conditions or situations that an object or system can be in.
- Each state is represented by a rectangle with a name.
- Transitions: These are the movements or changes from one state to another in response to an event or condition. They are represented by arrows with labels that indicate the trigger for the transition.
- Events: These are the occurrences or stimuli that cause a transition from one state to another. They are represented by small circles along the arrows.
- Actions: These are the operations or activities that occur when a transition takes place. They are represented by labels on the arrows or on the state itself.
- Guards: These are the conditions that must be true for a transition to occur. They are represented by labels on the arrows or on the state itself.
- Start and End Points: These indicate the beginning and end of the state chart diagram. The start point is represented by a filled circle and the end point is represented by a filled circle with a ring around it.



UML STATECHART DIAGRAM FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML state chart diagram for an order system of application

EXPERIMENT-6 UML ACTIVITY DIAGRAM

AIM: To draw a UML activity diagram for an order system of application using Star UML.

DESCRIPTION:
An activity diagram is a type of UML (Unified Modeling Language) diagram used to model the flow of activities or actions within a system or process. It represents the steps or activities that are required to achieve a particular goal, and shows the flow of control between them.

PURPOSE:
The purpose of an activity diagram is to model the flow of activities or actions that make up a process, workflow, or use case in a system. It is a type of behavioral diagram that shows the sequential and parallel activities that need to be performed in order to achieve a particular goal or outcome.
Activity diagrams can be used to model a wide range of scenarios, including business processes, software workflows, and system use cases. They are particularly useful for modeling complex or multi-step processes, as they provide a visual representation of the steps involved and the relationships between them.

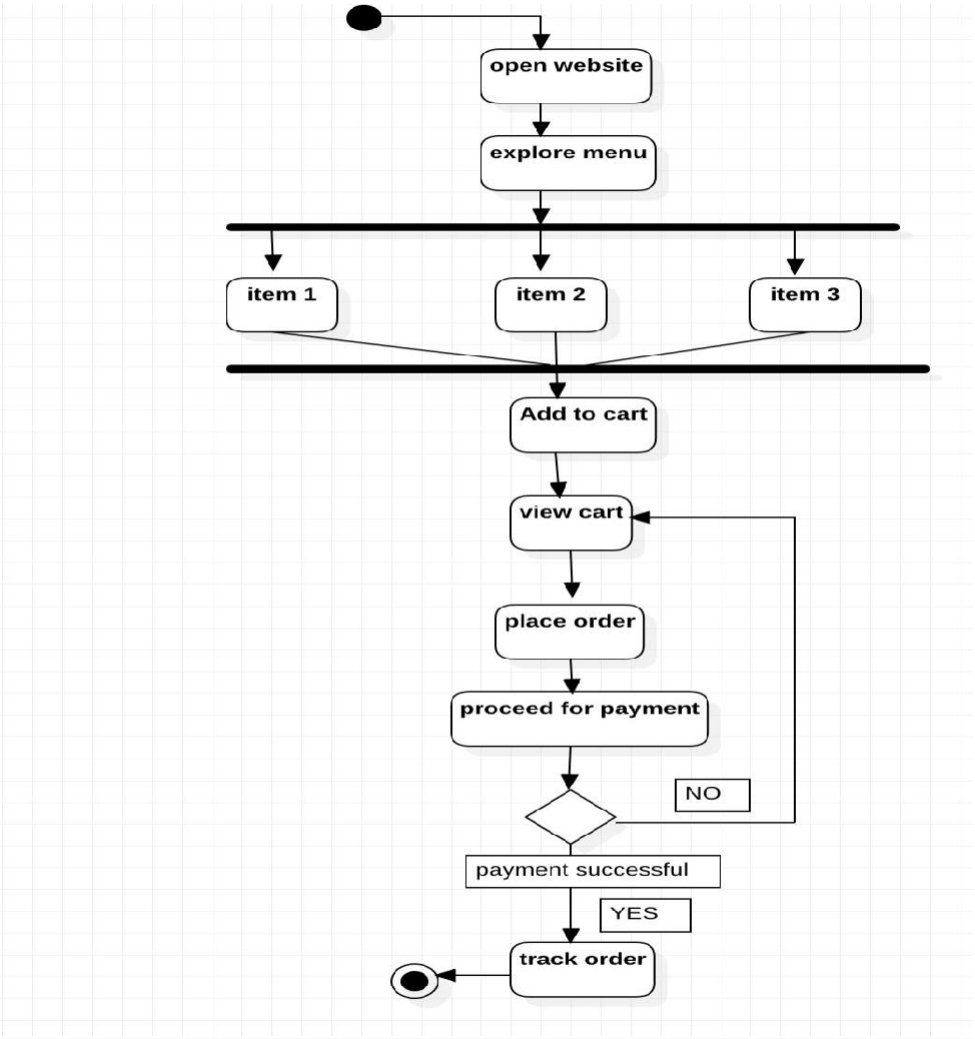
- Clarifying requirements
- Improving communication
- Supporting analysis and design.
- Supporting testing and debugging

HOW TO DRAW:

- Identify the purpose and scope of the activity diagram. Determine what process or workflow you want to model and what the specific goals of the diagram are.
- Identify the actors and objects involved in the process or workflow. Actors are external entities that interact with the system, while objects are internal entities that perform activities.
- Create a start node and an end node for the diagram. The start node represents the beginning of the process or workflow, while the end node represents the final outcome or goal.
- Identify the different activities or actions that need to be performed in order to achieve the goal of the process. These can be represented as nodes in the diagram.
- Connect the nodes with arrows to show the flow of activities. Use decision nodes to represent conditional branches in the workflow, and merge nodes to represent the merging of multiple branches.
- Add swimlanes to the diagram to represent different roles or departments involved in the process.
- Use annotations and notes to provide additional information or clarification on the activities or nodes.
- Review and refine the diagram to ensure that it accurately represents the process or workflow and achieves the intended purpose.

The main components of an activity diagram are:

- Activities: These are the specific actions or tasks that need to be performed. They are represented by rounded rectangles with descriptive labels inside.
- Control Flow: This is the directional flow of the activities, represented by arrows or lines that connect them. The arrows show the sequence in which the activities are performed. Decisions: These are points in the process where the flow can diverge based on a certain condition. They are represented by diamond shapes with labels that describe the condition. Merges: These are points where different paths in the process converge back into a single flow. They are represented by diamond shapes with multiple incoming arrows and a single outgoing arrow.
- Forks: These are points where the process splits into multiple concurrent paths. They are represented by a vertical bar with multiple outgoing arrows.
- Joins: These are points where the concurrent paths merge back into a single flow. They are represented by a vertical bar with multiple incoming arrows.
- Swimlanes: These are used to organize the activities based on the roles or responsibilities of different individuals or groups. They are represented by columns or rows with labels at the top.
- Together, these components form a visual representation of the steps or activities required to achieve a specific goal, and show how the different activities are connected and flow into each other.



UML ACTIVITY DIAGRAM FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML activity diagram for an order system of application

EXPERIMENT-7 UML USE-CASE AND CLASS DIAGRAM

AIM: To draw a UML use case diagram for an order system of application using Star UML.

DESCRIPTION:

A use case diagram is a visual representation of the functional requirements of a system. It is a type of UML (Unified Modelling Language) diagram that shows the actors (users) and use cases (functionality) of a system and how they interact with each other. The main purpose of a use case diagram is to describe the functional requirements of a system in a clear and concise way.

PURPOSE:

The purpose of a use case diagram is to provide a high-level view of the functional requirements of a system or application, and to illustrate the interactions between actors (users, systems, or external entities) and the system. Some of the key purposes of a use case diagram include:

- Modelling system behaviour
- Requirements gathering
- Communication tool
- Testing and validation

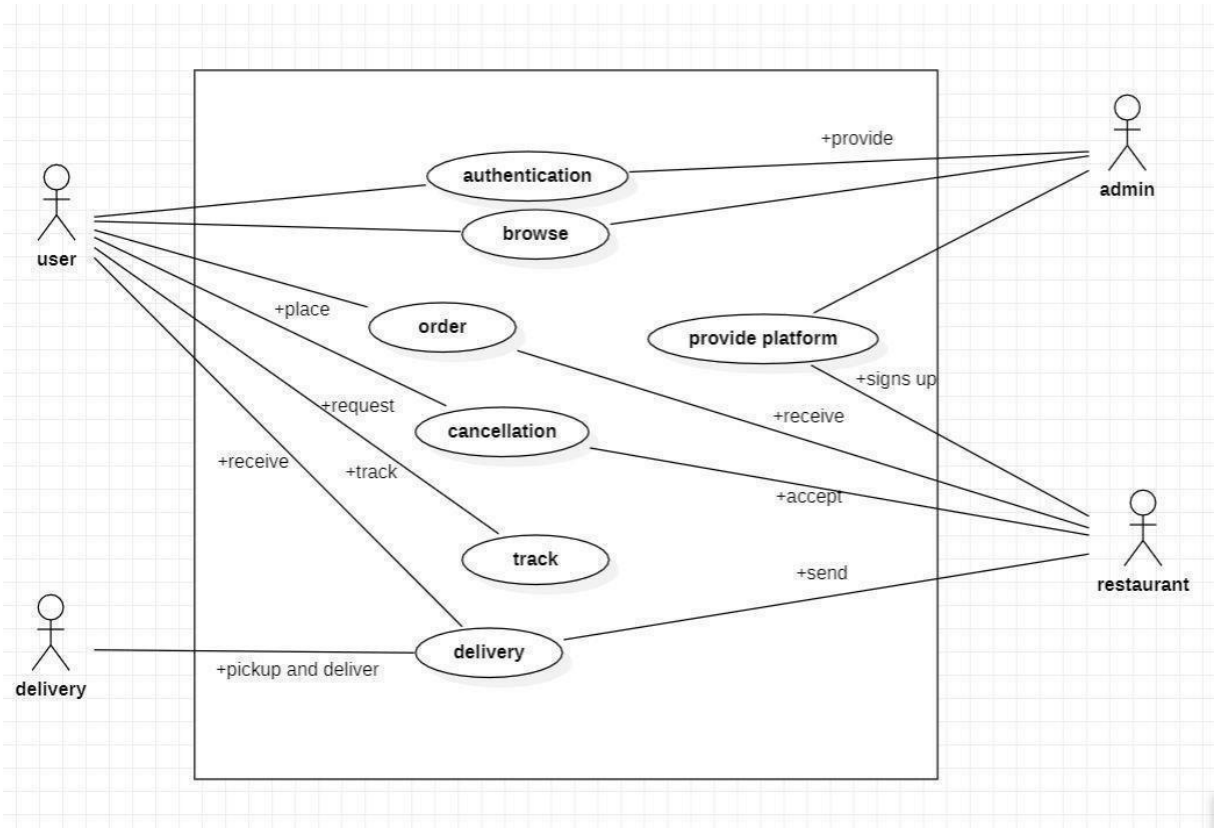
HOW TO DRAW:

- Identify the system boundaries: The first step is to identify the boundary of the system being modelled. This involves identifying the actors that interact with the system and defining the scope of the system.
- Identify the actors: The next step is to identify the actors that interact with the system. Actors can be users, external systems, or other entities that interact with the system.
- Identify the use cases: The next step is to identify the use cases that the system must support. Use cases represent the functionality provided by the system to the actors.
- Connect the actors and use cases: Use lines to connect the actors and use cases in the diagram. These lines represent the interactions between the actors and the use cases. Use cases can be associated with one or more actors.
- Include or extend relationships: If some use cases are related to other use cases, you can indicate this using the include or extend relationship. The include relationship indicates that one use case includes the functionality of another use case. The extend relationship indicates that one use case extends the functionality of another use case.
- Refine the use case diagram: Once you have the basic structure of the use case diagram in place, you can refine it by adding more detail. This might involve adding additional actors or use cases, or adding more information to existing use cases.

- Review and revise: Finally, review the use case diagram to ensure that it accurately represents the system and that all requirements are captured. Revise the diagram as necessary to ensure that it is complete and accurate.

The main components of a use case diagram include:

- Actors: Actors are the users or external systems that interact with the system being modelled. Actors are represented by stick figures on the diagram.
- Use Cases: Use cases are the specific functions or tasks that the system performs. Use cases are represented by ovals on the diagram.
- Relationships: Relationships describe how actors and use cases interact with each other. There are three types of relationships in a use case diagram:
- Association: An association is a connection between an actor and a use case. It shows that the actor is involved in the use case.
- Extend: An extend relationship shows that a use case can be extended with additional functionality. This is represented by an arrow with an open arrowhead.
- Include: An include relationship shows that a use case includes another use case as a step in its process. This is represented by an arrow with a closed arrowhead.
- System Boundary: The system boundary is a rectangle that surrounds the use cases and actors in the diagram. It represents the boundary of the system being modelled.



USE-CASE DIAGRAM FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML use case diagram for an order system of application.

UML CLASS DIAGRAM

AIM: To draw a UML class diagram for an order system of application using Star UML.

DESCRIPTION:

A class diagram is a type of UML (Unified Modelling Language) diagram that describes the structure of a system by showing the classes in the system and the relationships between them. A class is a blueprint for creating objects that have properties and methods, and the relationships between classes describe how the objects interact with each other.

PURPOSE:

The purpose of a class diagram is to provide a high-level view of the structure and behavior of a system or application, and to illustrate the relationships between classes and objects within the system. Some of the key purposes of a class diagram include:

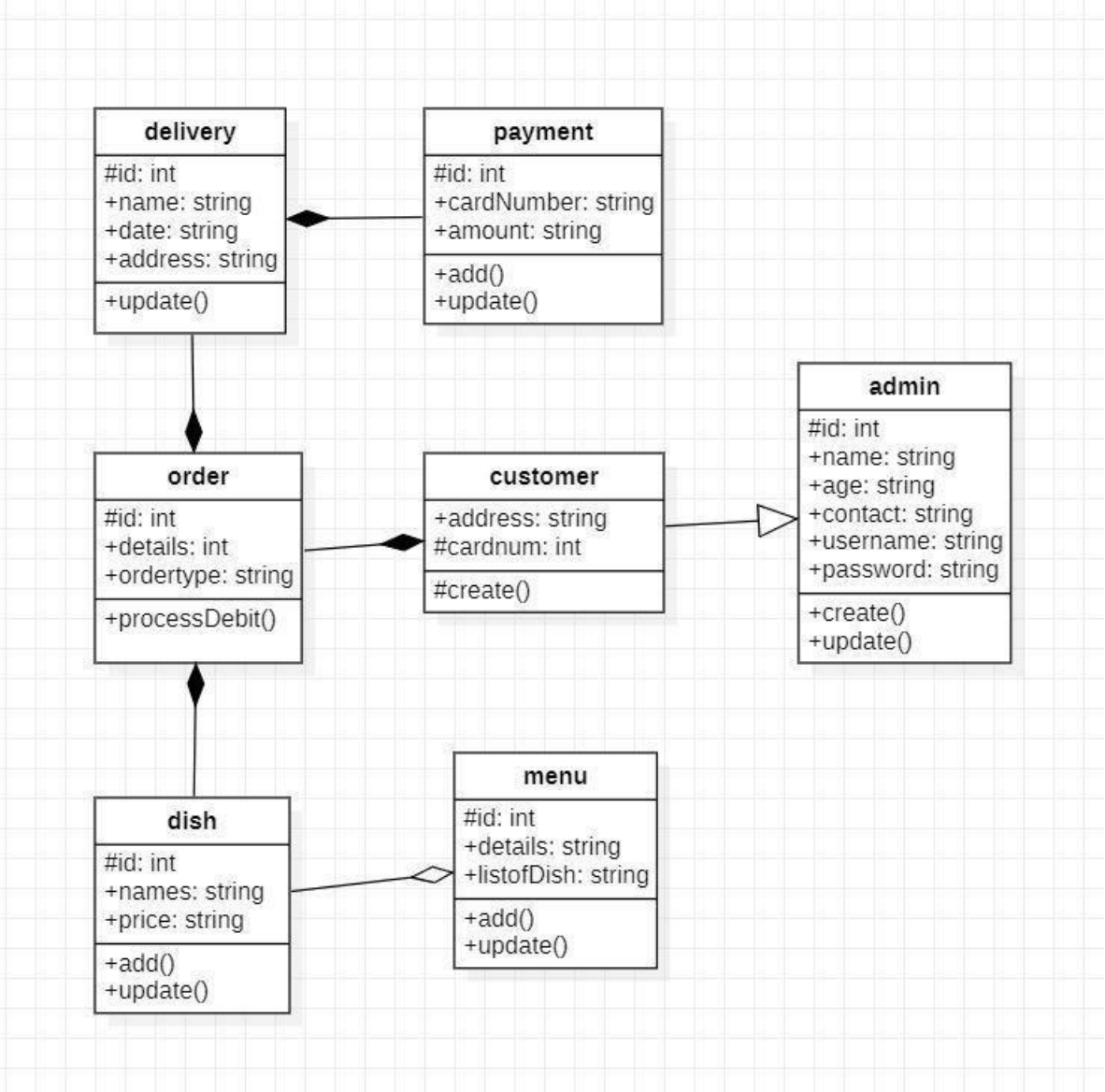
- Modeling system structure
- Visualizing system behavior
- Communication tool
- Code generation

HOW TO DRAW:

- Identify classes: The first step is to identify the classes that are required to model the system or application. This involves identifying the different objects or entities that are part of the system, and grouping them into classes based on their characteristics and behavior.
- Identify attributes: Once the classes have been identified, the next step is to identify the attributes that are associated with each class. Attributes describe the characteristics or properties of a class, such as its name, size, or color.
- Identify methods: After identifying the attributes, the next step is to identify the methods that are associated with each class. Methods describe the behavior or actions that a class can perform, such as calculating a value or updating a database.
- Define relationships: Once the classes, attributes, and methods have been identified, the next step is to define the relationships between the classes. Relationships describe the way that one class is connected to another class, and can include inheritance, composition, aggregation, and association.
- Draw the diagram: The final step is to draw the class diagram, using UML notation to represent the classes, attributes, methods, and relationships. The class diagram should be easy to read and understand, and should accurately represent the structure and behavior of the system or application being modeled.

The main components of a class diagram include:

- Classes: Classes are the building blocks of a class diagram. They represent a template for creating objects with common properties and behaviors. Classes are represented as rectangles with the class name written inside.
- Attributes: Attributes are the properties of a class. They describe the characteristics of the objects created from the class. Attributes are represented as lines with the name of the attribute and its data type.
- Methods: Methods are the behaviors of a class. They define what the objects created from the class can do. Methods are represented as lines with the name of the method and its parameters and return type.



UML CLASS DIAGRAM FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML class diagram for an order system of application.

EXPERIMENT-8 UML COMPONENT DIAGRAM

AIM: To draw a UML component diagram for an order system of application using Star UML.

DESCRIPTION:

A component diagram is a type of UML diagram that shows the structural relationship between the components in a system. It is used to model the physical components of a system and the dependencies among them.

PURPOSE:

The purpose of a component diagram in UML is to model the physical components of a system or application, and to show how they interact with each other to create the overall system. The main purposes of a component diagram include:

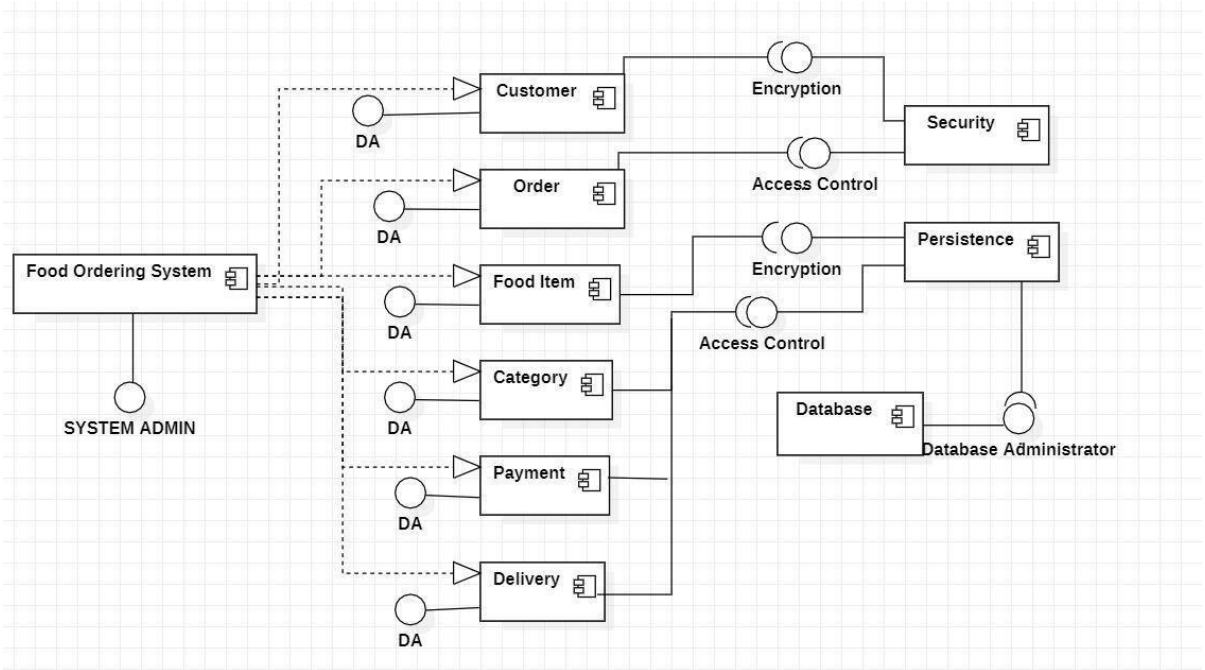
- Component visualization
- System design
- System maintenance
- Collaboration
- Testing

HOW TO DRAW:

- Identify the components: The first step is to identify the different components of the system or application that you want to model. These may include software components such as classes, modules, or libraries, or physical components such as servers, databases, or hardware devices.
- Determine the relationships: Once you have identified the components, the next step is to determine how they are related to each other. This may include dependencies, associations, or interfaces between components.
- Draw the components: The next step is to draw the components on the diagram, using the appropriate UML notation. For software components, this may include using class symbols or interface symbols, while physical components may be represented by cloud symbols or server symbols.
- Connect the components: Once you have drawn the components, the next step is to connect them with the appropriate relationships. This may include using dependency arrows, association lines, or interface symbols.
- Label the diagram: Finally, you should label the diagram with appropriate names, descriptions, and other information to help users understand the purpose and functionality of each component.

The main symbols used in a component diagram are:

- Component: It is represented as a rectangle with two parts, one part containing the name of the component and the other containing the keyword “component”. A component is a modular, deployable and replaceable part of a system that encapsulates its implementation and exposes a set of interfaces.
- Interface: It is represented as a small circle on the edge of a component. An interface defines a set of operations that a component provides or requires from other components.
- Dependency: It is represented as an arrow between two components, indicating that one component depends on another component. A dependency can be either a requirement or a
- realization. A requirement is a dependency in which one component needs the services of another component to function properly. A realization is a dependency in which one component implements the services provided by another component.
- Port: It is represented as a small square on the edge of a component. A port is a point of interaction between a component and its environment.
- Connector: It is represented as a line between two ports or interfaces, indicating that they are connected. A connector can be either a delegation or an assembly. A delegation is a connector in which one component delegates some of its operations to another component. An assembly is a connector in which one component is composed of several sub-components.
- Package: It is represented as a folder-like icon, used to group related components together.



UML COMPONENT DIAGRAM FOR ONLINE FOOD SERVICES

RESULT: Thus, we have successfully drawn the UML component diagram for an order system of application.

EXPERIMENT-9 UML INTERACTION DIAGRAM

AIM: To draw a UML interaction diagram for an order system of application using Star UML.

DESCRIPTION:

An interaction diagram is a graphical representation of the interactions that take place between objects in a system or between actors in a use case. It shows how objects or actors collaborate with each other to achieve a specific goal or complete a certain task. Interaction diagrams are a part of the Unified Modelling Language (UML) and are commonly used during the design phase of software development to visualize and communicate the behaviour of a system.

PURPOSE:

The purpose of interaction diagrams in UML is to model the dynamic behavior of a system or application by showing how objects interact with each other over time. Interaction diagrams provide a graphical representation of the flow of messages and actions between objects, and they can help developers better understand the logic and behavior of a system.

There are two types of interaction diagrams in UML: sequence diagrams and communication diagrams. Sequence diagrams focus on the time-based sequence of messages exchanged between objects or actors, while communication diagrams illustrate the interactions as a set of objects and the messages that they exchange.

Interaction diagrams help to visualize and analyse the behaviour of a system, identify potential problems or bottlenecks, and improve communication among team members. They are also useful for testing and verifying system behaviour before implementation.

The main purposes of interaction diagrams include:

- System design: Interaction diagrams are used in the early stages of system design to model the behavior of a system and to identify potential issues or problems with the system's design.
- System testing: Interaction diagrams can also be used to guide system testing, by providing a visual representation of the interactions between objects in a system and helping to ensure that all possible interactions have been tested.
- Communication: Interaction diagrams can be used as a communication tool between developers, project managers, and other stakeholders, to help everyone understand how the system works and to ensure that everyone has a common understanding of the system's behavior.
- Documentation: Interaction diagrams can be used as part of system documentation, to provide a visual representation of the system's behavior and to help future developers better understand how the system works.

HOW TO DRAW:

Sequence Diagram:

- Identify the objects: The first step is to identify the objects that will be included in the sequence diagram. These are typically the objects that are involved in the interaction that you want to model.
- Identify the messages: Next, you need to identify the messages that are exchanged between the objects. These messages represent the actions that the objects perform during the interaction.
- Draw the lifelines: Each object in the sequence diagram is represented by a vertical line called a "lifeline". Draw a lifeline for each object, and label each lifeline with the name of the corresponding object.
- Add the messages: Draw horizontal arrows between the lifelines to represent the messages that are exchanged between the objects. Label each arrow with the name of the message.

The components of each type of interaction diagram are as follows:

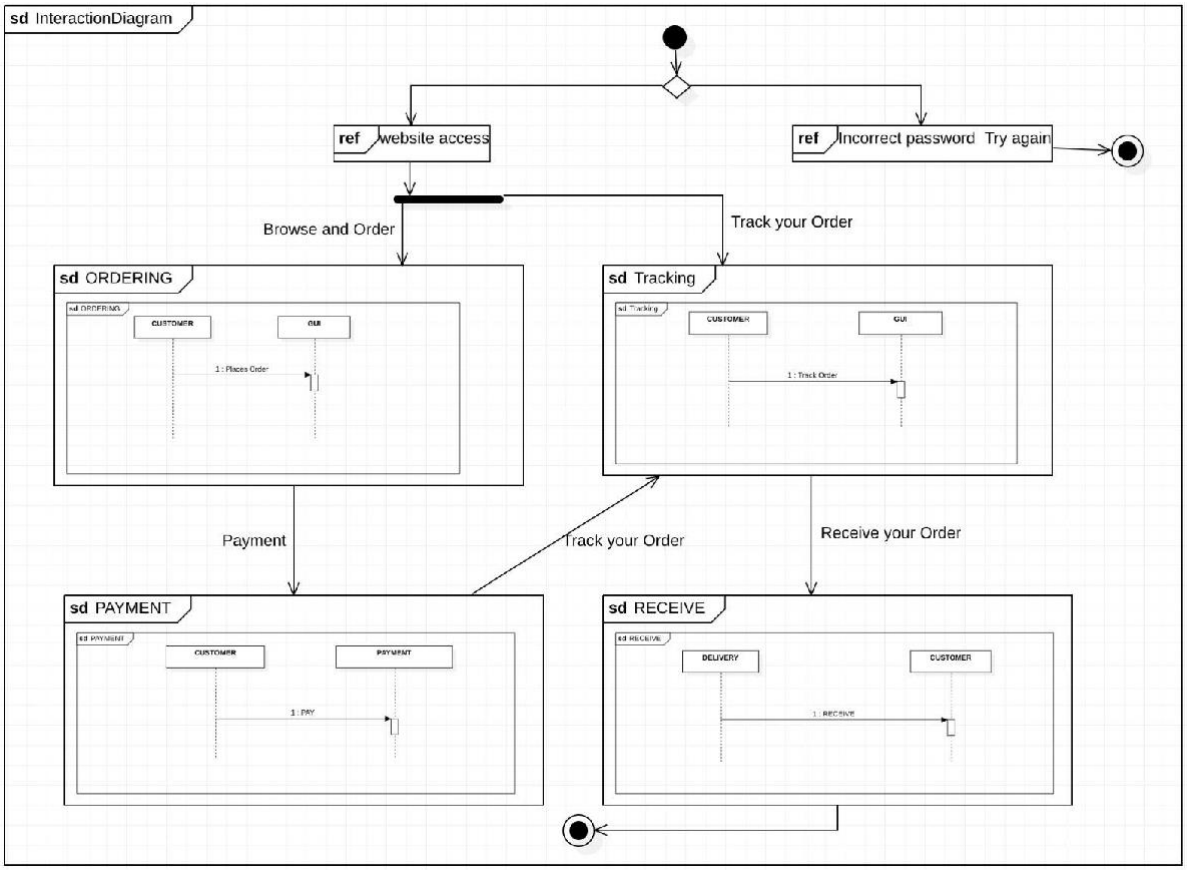
- Lifeline: A vertical line that represents an object or participant involved in the sequence of actions in the system.
- Execution Occurrence: A rectangle that represents the time interval during which an object performs a particular action.
- Message: A horizontal arrow that represents a communication between two lifelines, indicating the direction of the message and its name.
- Activation Bar: A horizontal bar located on a lifeline, indicating the period of time during which the object is busy executing a task.

Collaboration Diagram:

- Identify the objects: The first step is to identify the objects that will be included in the collaboration diagram. These are typically the objects that are involved in the interaction that you want to model.
- Draw the objects: Draw a box for each object, and label each box with the name of the corresponding object.
- Add the messages: Draw lines between the boxes to represent the messages that are exchanged between the objects. Label each line with the name of the message.

The components of each type of interaction diagram are as follows:

- Object: A rectangular box that represents an object or participant involved in the collaboration.
- Link: A line that connects two objects and represents a communication or association between them.
- Message: A numbered arrow that represents the order and direction of messages being sent between objects.
- Multiplicity: A notation that represents the number of instances of an object that participate in the collaboration.
- Role Name: A name assigned to an object to help identify its purpose or role in the collaboration.



UML INTERACTION FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML interaction diagram for an order system of application

EXPERIMENT-10 UML COLLABORATION DIAGRAM

AIM: To draw a UML collaboration diagram for an order system of application using Star UML.

DESCRIPTION:

A collaboration diagram is a type of interaction diagram in UML (Unified Modeling Language) that depicts the interactions and relationships among objects participating in a particular scenario or use case.

It visually represents the interactions between objects and how they collaborate to accomplish a specific task or functionality. Collaboration diagrams are also known as communication diagrams as they show how objects communicate with each other to achieve a common goal.

PURPOSE:

The purpose of a collaboration diagram in UML is to illustrate the interactions and relationships between objects or components in a system. It shows the various objects or components involved in a particular scenario and how they communicate with each other to achieve a common goal.

The collaboration diagram is often used to model the dynamic behavior of a system, showing how objects collaborate with each other in order to complete a particular task or scenario. It allows developers and stakeholders to visualize the flow of control and communication between different objects or components, which can help them to identify potential issues and improve the overall design of the system.

In a collaboration diagram, the objects are represented as boxes, and the messages exchanged between them are represented as arrows. The sequence of messages is shown by numbering the arrows. Additionally, the lifelines of the objects are shown, which represents the time period during which the object is active or involved in the collaboration.

Collaboration diagrams are useful in understanding complex systems, analysing object interactions, identifying communication gaps, and for designing and testing software applications. They provide a clear picture of the objects involved in the system and how they interact with each other, making them an important tool for software developers and system architects.

HOW TO DRAW:

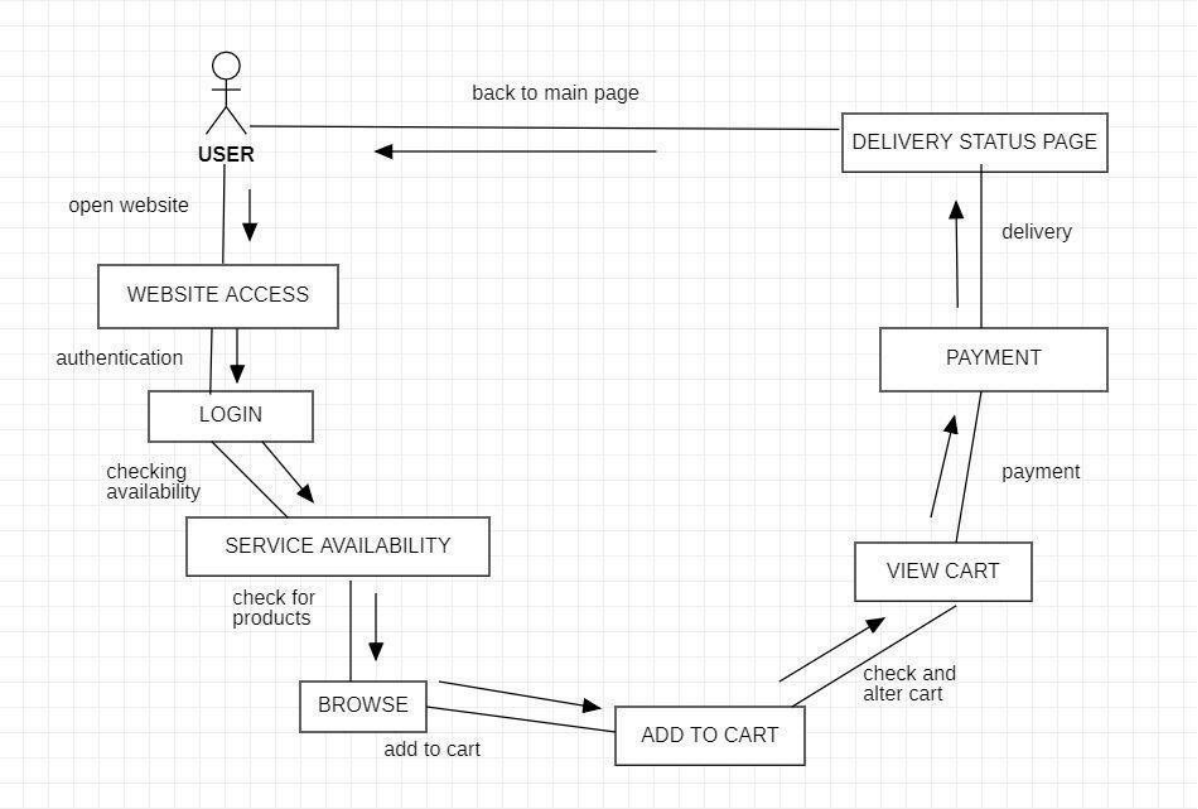
Identify the objects: The first step is to identify the objects that will be included in the collaboration diagram. These are typically the objects that are involved in the interaction that you want to model.

Draw the objects: Draw a box for each object, and label each box with the name of the corresponding object.

Add the messages: Draw lines between the boxes to represent the messages that are exchanged between the objects. Label each line with the name of the message.

The components of each type of interaction diagram are as follows:

- Object: A rectangular box that represents an object or participant involved in the collaboration.
- Link: A line that connects two objects and represents a communication or association between them.
- Message: A numbered arrow that represents the order and direction of messages being sent between objects.
- Multiplicity: A notation that represents the number of instances of an object that participate in the collaboration.
- Role Name: A name assigned to an object to help identify its purpose or role in the collaboration.



UML COLLABORATION DIAGRAM FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML collaboration diagram for an order system of application.

EXPERIMENT-11 UML DEPLOYMENT DIAGRAM

AIM: To draw a UML deployment diagram for an order system of application using Star UML.

DESCRIPTION:

The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships. It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.

PURPOSE:

The purpose of deployment diagrams can be described as −

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components. Describe the runtime processing nodes.

HOW TO DRAW:

- Identify the software components and hardware nodes: Start by identifying the software components (such as applications, databases, middleware, etc.) and hardware nodes (such as servers, routers, switches, etc.) that will be involved in the deployment.
- Draw the nodes: Use rectangles to represent the hardware nodes and place them on the diagram.
- Add the components: Use component symbols (rectangles with a smaller rectangle inside) to represent the software components and place them on the diagram. Label each component with a descriptive name and include any necessary details about its functionality.
- Add the deployment relationships: Use lines to show the deployment relationships between the components and the nodes. These relationships can be either direct (when a component is deployed on a specific node) or indirect (when a component is deployed on multiple nodes or on a cluster).

Components of deployment diagram:

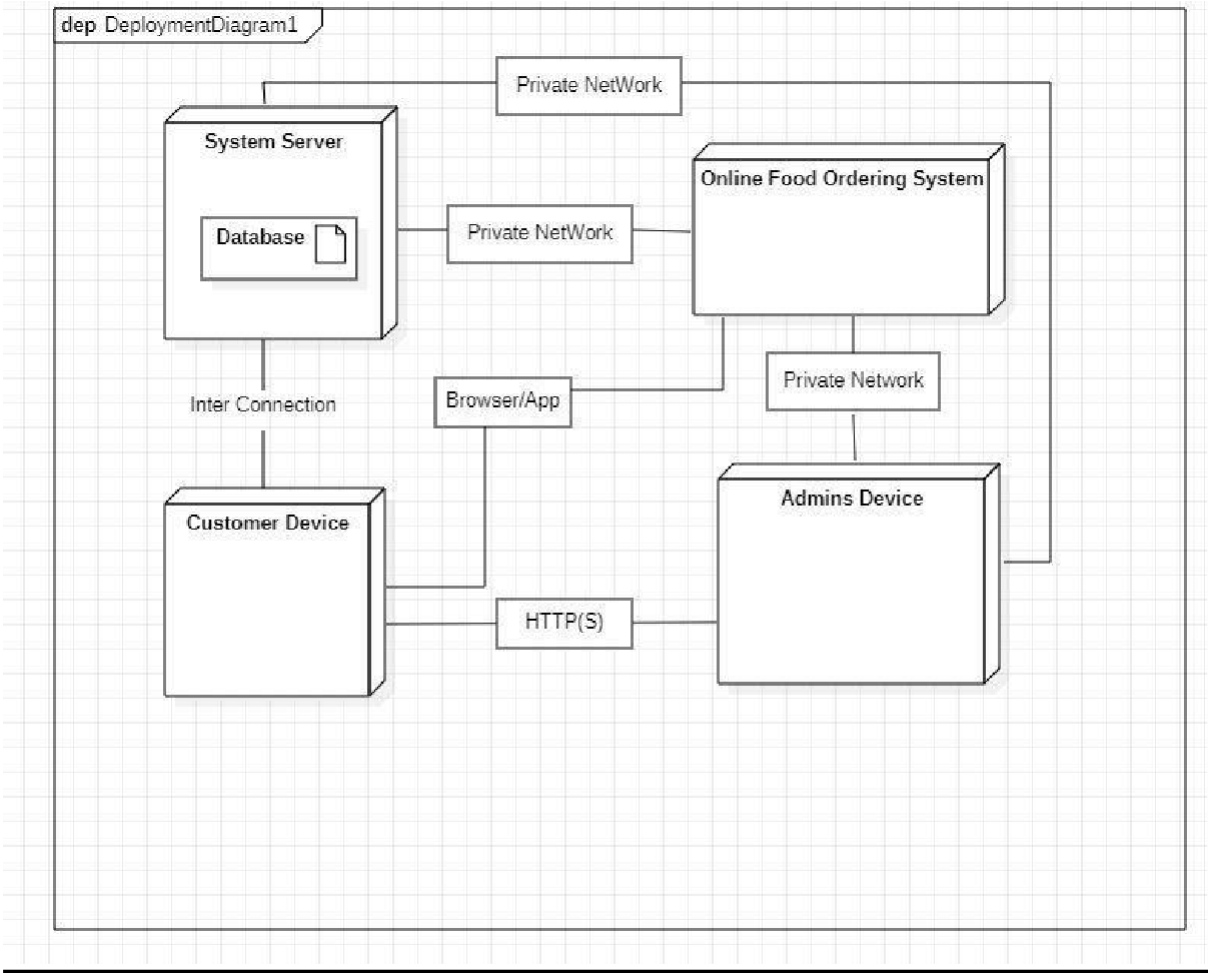
Nodes: Nodes represent the physical hardware devices or software environments on which the system components are deployed.

Components: Components represent the software artifacts that are deployed to the nodes.

Artifacts: Artifacts are the physical files, libraries, or other software elements that make up a component

Deployment Relationships: Deployment relationships show how the components are deployed onto the nodes. These relationships include associations, dependencies, and generalizations.

Communication Paths: Communication paths show how nodes communicate with each other over the network.



UML DEPLOYMENT DIAGRAM FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML DEPLOYMENT diagram for an order system of application

EXPERIMENT-12 UML SEQUENCE DIAGRAM WITH THREADS

AIM: To draw a UML sequence diagram with threads for an order system of application using Star UML.

DESCRIPTION:

In a sequence diagram, a thread represents a separate flow of execution or control within the system being modelled. Each thread typically corresponds to a separate unit of functionality or process within the system and can interact with other threads and objects in the system through method calls and message exchanges. Threads are often used in sequence diagrams to illustrate concurrent or parallel processing, where multiple threads are executing simultaneously and interacting with each other and the rest of the system. Each thread is represented as a separate vertical line on the diagram, with arrows or messages indicating the flow of control and communication between threads and objects.

PURPOSE:

Threads are used in sequence diagrams to represent concurrent or parallel execution of different parts of a system. The purpose of using threads in sequence diagrams is to illustrate how multiple threads of control interact with each other and with the system components to achieve a particular task or scenario.

In a sequence diagram, each thread is represented by a vertical line that runs through the diagram. Each message or action that is associated with a particular thread is shown as a horizontal arrow that crosses the thread line. This allows you to see how different threads interact with each other over time, and how they exchange data and signals to achieve the desired behavior.

The use of threads in sequence diagrams is particularly useful for systems that rely heavily on parallel processing, such as multi-core processors or distributed systems. By using threads, you can visualize how different parts of the system interact with each other in real-time, allowing you to identify potential bottlenecks or performance issues.

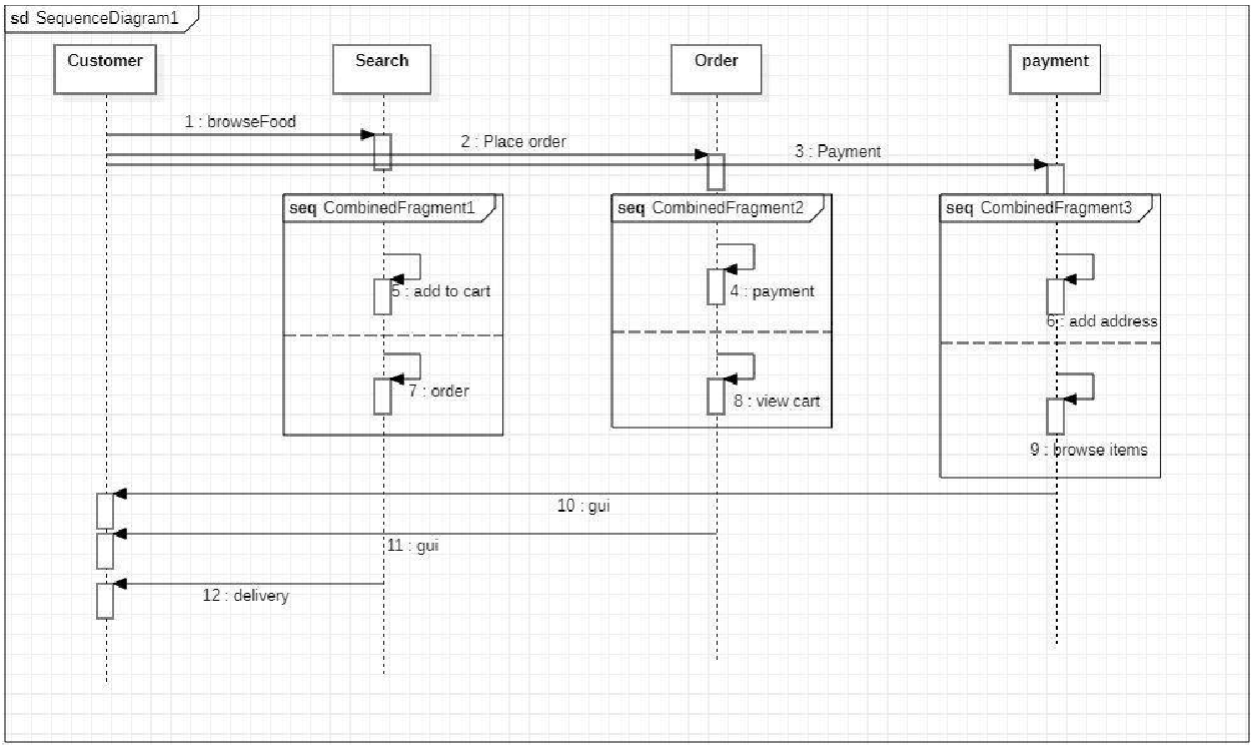
In addition, the use of threads in sequence diagrams can help to clarify complex scenarios that involve multiple actors and interactions, allowing developers and stakeholders to better understand the system behavior and identify potential issues early in the development process.

HOW TO DRAW:

- Identify the different threads that are involved in the scenario you are modeling. For example, if you are modeling a banking system, you may have separate threads for the customer, the teller, and the ATM.
- Draw a vertical line for each thread, running from the top of the diagram to the bottom.
- Identify the messages or actions that are associated with each thread. For example, if the customer wants to withdraw money from their account, they may send a message to the teller thread, which will in turn send a message to the banking system thread.
- Draw horizontal arrows that cross the thread lines to represent each message or action. Label each arrow with a description of the message or action.
- Use dotted lines to represent asynchronous messages or actions, which do not block the thread while they are being processed.
- Add any additional components, such as external systems, databases, or middleware, that are involved in the scenario.

The components of a sequence diagram can include:

- Objects: Objects represent the different entities or actors that are involved in the scenario. For example, in a banking system, you may have objects for the customer, the teller, and the ATM.
- Messages: Messages represent the communication between the different objects. Each message is labeled with a description of the action being performed.
- Activation boxes: Activation boxes represent the period of time during which an object is active and processing a message. The top and bottom of the box represent the start and end of the processing period.
- Lifelines: Lifelines represent the existence of an object over time. Each object is associated with a lifeline, which runs vertically through the diagram.
- Combined fragments: Combined fragments are used to represent alternative or parallel paths in the sequence diagram. They are represented by boxes with different types of markers, such as "alt" or "par".
- Constraints: Constraints are used to specify any conditions or limitations on the behavior of the system or objects in the scenario. They can be represented by annotations on the diagram.



UML SEQUENCE DIAGRAM WITH THREADS FOR ONLINE FOOD DELIVERY SERVICES

RESULT: Thus, we have successfully drawn the UML sequence diagram with threads for an order system of application.