



RAJ KUMAR GOEL INSTITUTE OF TECHNOLOGY

Approved by AICTE (Ministry of Education) & PCI (Ministry of Health & FW) GOI, Affiliated to Dr. APJ Abdul Kalam Technical University, Lucknow
AKTU College Code-033

Accredited by NAAC ('A' Grade), NBA Accredited Programs (B.Tech-ECE, IT) & B.Pharma



SOFTWARE ENGINEERING LAB FILE (BCS -651)

Name	
Roll No.	
Section-Batch	



AKTU College Code-033



NAAC Accredited with Grade A

[illegible]



VISION OF THE INSTITUTE

To continually develop excellent professionals capable of providing sustainable solutions to challenging problems in their fields and prove responsible global citizens.

MISSION OF THE INSTITUTE

We wish to serve the nation by becoming a reputed deemed university for providing value based professional education.

VISION OF THE DEPARTMENT

To be recognized globally for delivering high quality education in the ever changing field of computer science & engineering, both of value & relevance to the communities we serve.

MISSION OF THE DEPARTMENT

1. To provide quality education in both the theoretical and applied foundations of Computer Science and train students to effectively apply this education to solve real world problems.
2. To amplify their potential for lifelong high quality careers and give them a competitive advantage in the challenging global work environment.

COURSE OUTCOMES (COs)

- CO1:** Identify ambiguities, inconsistencies and incompleteness from a requirements specification and state functional and non-functional requirement
- CO2:** Identify different actors and use cases from a given problem statement and draw use case diagram to associate use cases with different types of relationship
- CO3:** Draw a class diagram after identifying classes and association among them
- CO4:** Graphically represent various UML diagrams, and associations among them and identify the logical sequence of activities undergoing in a system, and represent them pictorially
- CO5:** Able to use modern engineering tools for specification, design, implementation and testing



PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective



presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: The ability to use standard practices and suitable programming environment to develop software solutions.

PSO2: The ability to employ latest computer languages and platforms in creating innovative career opportunities.



EXPERIMENT – 1

Aim: Prepare a SRS document in line with the IEEE recommended standards for a given problem.

Creating a Software Requirements Specification (SRS) document for the Flappy Bird Game in accordance with the IEEE recommended standards involves structuring the document in a standardized format that ensures clarity, comprehensiveness, and consistency. Below is an outline and a basic example of how an SRS document for the Flappy Bird Game can be structured:

Software Requirements Specification (SRS) for Flappy Bird Game :

1. Introduction :

1.1 Purpose

The purpose of this document is to specify the software requirements for the Flappy Bird Game, a simple yet addictive mobile game where the player controls a bird to fly through gaps in pipes without hitting them.

1.2 Scope

This document covers the functional and non-functional requirements for the Flappy Bird Game. The game is designed for mobile platforms (iOS, Android) and will have the following features:

- Control of a bird character that flies upwards when the player taps the screen.
- Dynamic obstacles (pipes) that appear and move across the screen.
- Scoring system that increases when the bird successfully passes through the gap between pipes.
- A simple and engaging user interface.

1.3 Definitions, Acronyms, and Abbreviations

- Bird: The controllable character in the game.
- Pipe: Obstacles that the bird must navigate through.
- Gap: The space between two pipes where the bird must pass through.
- Tap: The action of tapping the screen to make the bird fly.
- Game Over: When the bird collides with a pipe or hits the ground.

1.4 References

- IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications.



- Flappy Bird: https://en.wikipedia.org/wiki/Flappy_Bird

1.5 Overview

The remainder of this document provides detailed functional and non-functional requirements, system models, and system constraints for the Flappy Bird Game.

2. Overall Description

2.1 Product Perspective

The Flappy Bird Game is a stand-alone mobile application that is part of the genre of casual mobile games. The game does not integrate with other systems but will function independently.

2.2 Product Features

The game will consist of the following major features:

- Bird Control: The player taps the screen to make the bird fly upward.
- Obstacle Generation: Pipes will appear at random intervals and move across the screen.
- Collision Detection: If the bird collides with pipes or the ground, the game ends.
- Scoring System: The player earns points by passing through the gaps between pipes.
- Game Restart: The player can restart the game after a game over.

2.3 User Classes and Characteristics

- Casual Gamers: Users who play the game for entertainment in short sessions.
- Hardcore Gamers: Users who aim to achieve high scores and master the game mechanics.

2.4 Operating Environment

- The game will be available on Android and iOS mobile platforms.
- Minimum hardware specifications:
 - Android: Version 4.1 (Jelly Bean) or higher.
 - iOS: Version 10.0 or higher.

2.5 Constraints

- The game should have minimal loading time.
- The game must be optimized for smooth performance, even on low-end devices.
- The game should not require an internet connection to play.



- The game's size must not exceed 50MB to ensure fast downloads.

2.6 Assumptions and Dependencies

- The game assumes that players are familiar with basic touch-screen controls.
- The game will depend on basic device sensors like touch input and gravity (for bird's fall speed).

3. Functional Requirements

3.1 Game Start

- FR1: The game shall start with a main menu that includes options like "Start", "High Scores", and "Exit".
- FR2: Upon selecting "Start", the game will begin with the bird positioned at the center of the screen.

3.2 Bird Control

- FR3: The bird shall fly upward each time the user taps the screen.
- FR4: The bird shall fall slowly when there is no screen tap.

3.3 Obstacles

- FR5: The game shall generate pipes at random vertical positions.
- FR6: The pipes will move from right to left.
- FR7: Each pair of pipes shall have a gap that the bird must pass through.

3.4 Collision Detection

- FR8: If the bird collides with a pipe or the ground, the game will end.
- FR9: The game will display a "Game Over" screen when the bird hits an obstacle.

3.5 Scoring System

- FR10: The score shall increase by 1 point each time the bird successfully passes through the gap between two pipes.
- FR11: The score will be displayed at the top of the screen during gameplay.



3.6 Game Restart

- FR12: After the game ends, the player will be able to restart the game by pressing the "Restart" button.
- FR13: The game will reset all variables and start from the beginning.

4. Non-Functional Requirements

4.1 Performance

- NFR1: The game shall maintain a smooth frame rate of at least 30 frames per second (FPS) on all supported devices.
- NFR2: The game shall have a response time of less than 100ms for all user interactions.

4.2 Usability

- NFR3: The game shall have a simple user interface that is intuitive and easy to navigate.
- NFR4: The game's instructions and menus shall be easy to understand, with clear call-to-action buttons.

4.3 Reliability

- NFR5: The game shall not crash or freeze during gameplay.
- NFR6: The game shall handle unexpected user interactions (such as tapping outside the game area) gracefully.

4.4 Maintainability

- NFR7: The game code shall be modular and well-documented to facilitate future updates.
- NFR8: The game shall be easy to update with new features (e.g., new obstacles or skins).

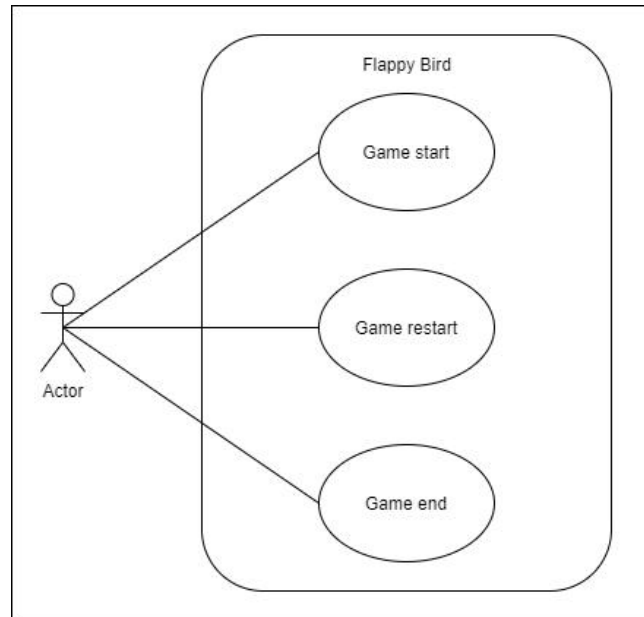
4.5 Security

- NFR9: The game shall not require any user personal data, ensuring user privacy is maintained.

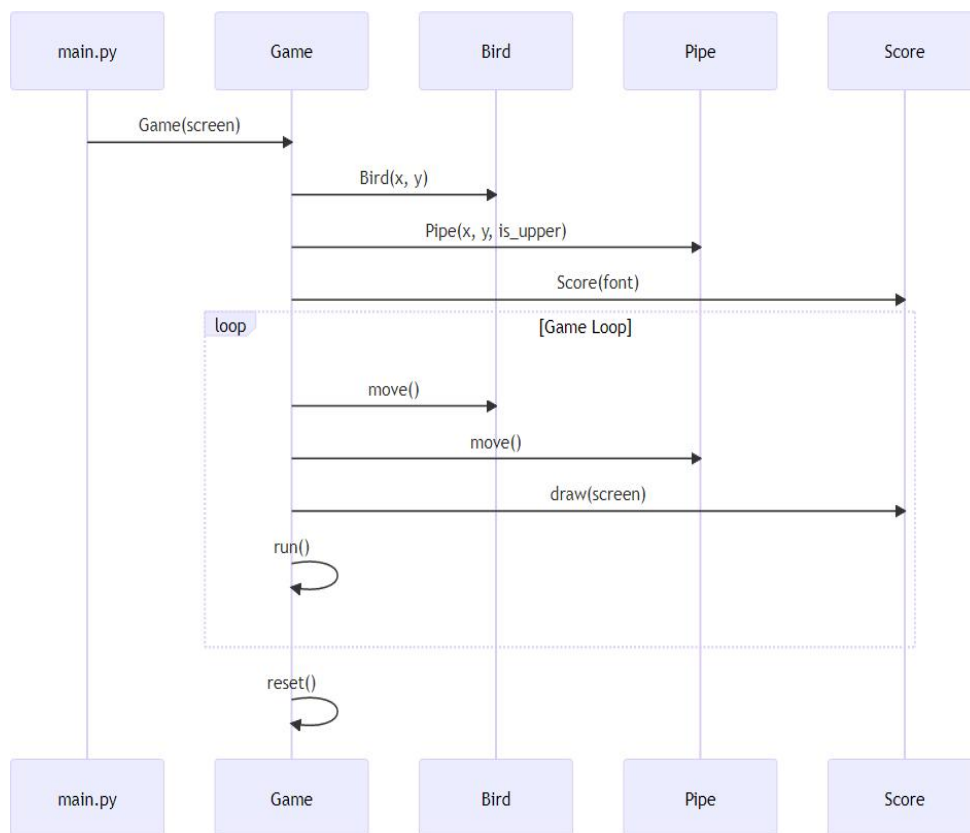
5. System Models



5.1 Use Case Diagram



5.2 Sequence Diagram





6. Other Requirements

- **6.1 Localization**: The game should support multiple languages, including English, Spanish, and French.

- **6.2 Accessibility**: The game should provide accessibility features like color contrast for the visually impaired.



EXPERIMENT - 2

Aim: Draw the use case diagram and specify the role of each of the actors Also state the precondition, post condition and function of each use case for a given problem.

Description:

Introduction: The Use Case Approach

- Ivar Jacobson & others introduced Use Case approach for elicitation & modelling.
- Use Case – give functional view. Use Cases are structured outline or template for the description of user requirements modelled in a structured language like English.
- Use case Scenarios are unstructured descriptions of user requirements.
- Use case diagrams are graphical representations that may be decomposed into further levels of abstraction.
- Actor: An actor or external agent, lies outside the system model, but interacts with it in some way.

Actor→Person, machine, information System

- Jacobson & others proposed a template for writing Use cases as shown below:

1. Introduction

Describe a quick background of the use case.

2. Actors

List the actors that interact and participate in the use cases.

3. Pre-Conditions

Pre-conditions that need to be satisfied for the use case to perform.

4. Post Conditions

Define the different states in which we expect the system to be in, after the use case executes.

5. Flow of events

5.1 Basic Flow

List the primary events that will occur when this use case is executed.

5.2 Alternative Flows

Any Subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow.

A use case can have many alternative flows as required.

6. Special Requirements

Business rules should be listed for basic & information flows as special requirements in the use case narration. These rules will also be used for writing test cases. Both success and failures scenarios should be described.

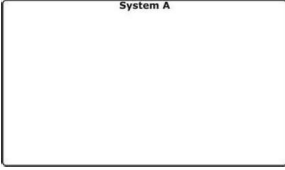
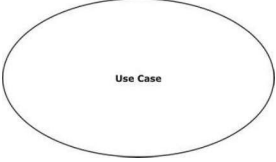
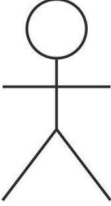
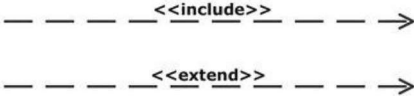


7. Use Case relationships

For Complex systems it is recommended to document the relationships between use cases. Listing the relationships between use cases also provides a mechanism for traceability.

Use Case Diagram Symbols

The most commonly used symbols for use case diagrams are as follows:

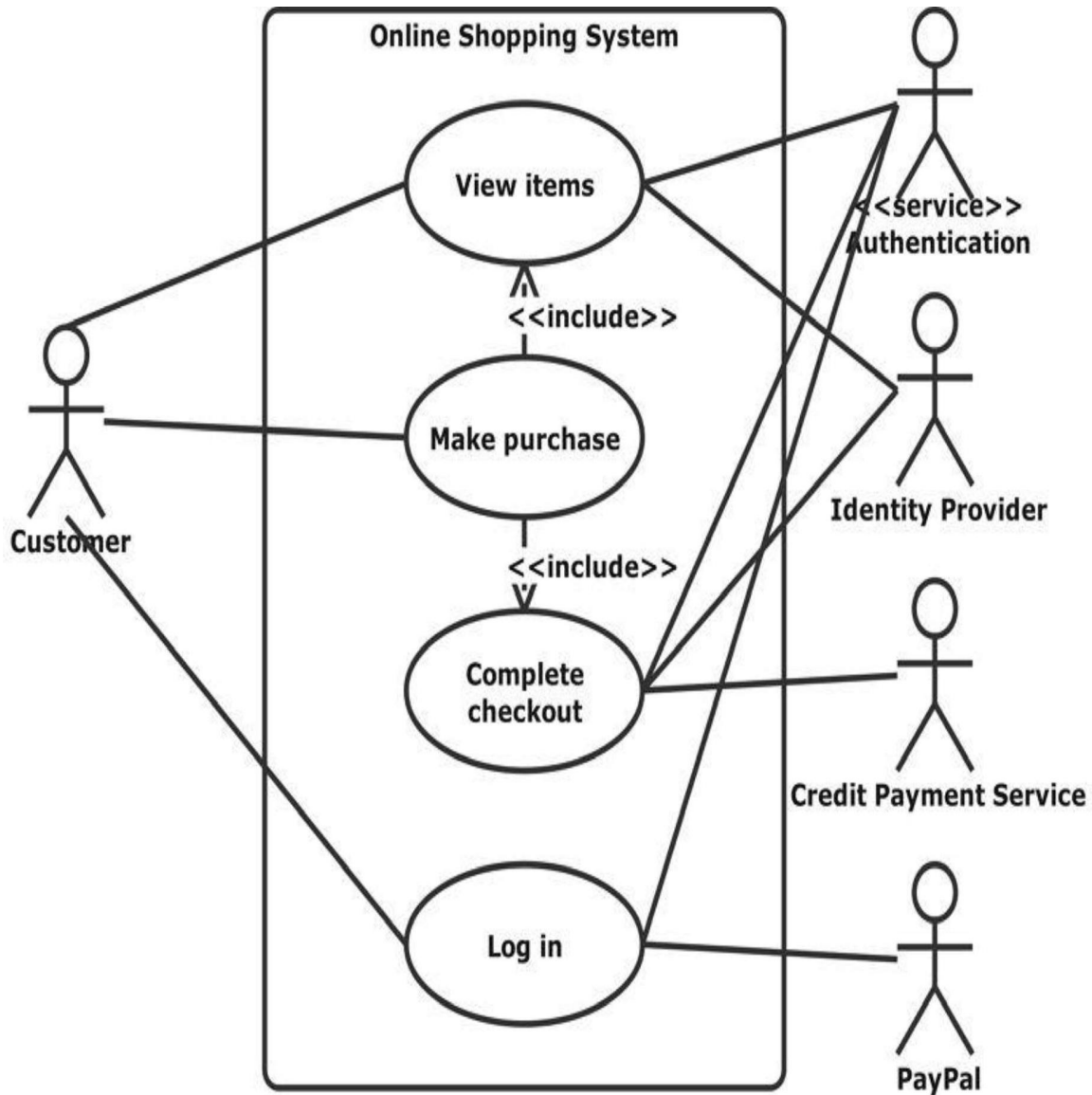
	System The rectangular boundary is the system. Use cases fall inside it, and actors will be placed outside it.
	Use Case An oval shape represents a use case. Use cases represent the functionality of the system, as well as the end-goal of the actor. Use cases should be placed inside the system.
	Actor When an actor interacts with the system, it triggers a use case. Actors should be placed outside the system.
	Relationships Arrows are used to indicate a relationship between an actor and a use case, or between two use cases. An extension indicates that one use case may include the behaviour of another use case. An inclusion represents one use case using the functionality of another use case.

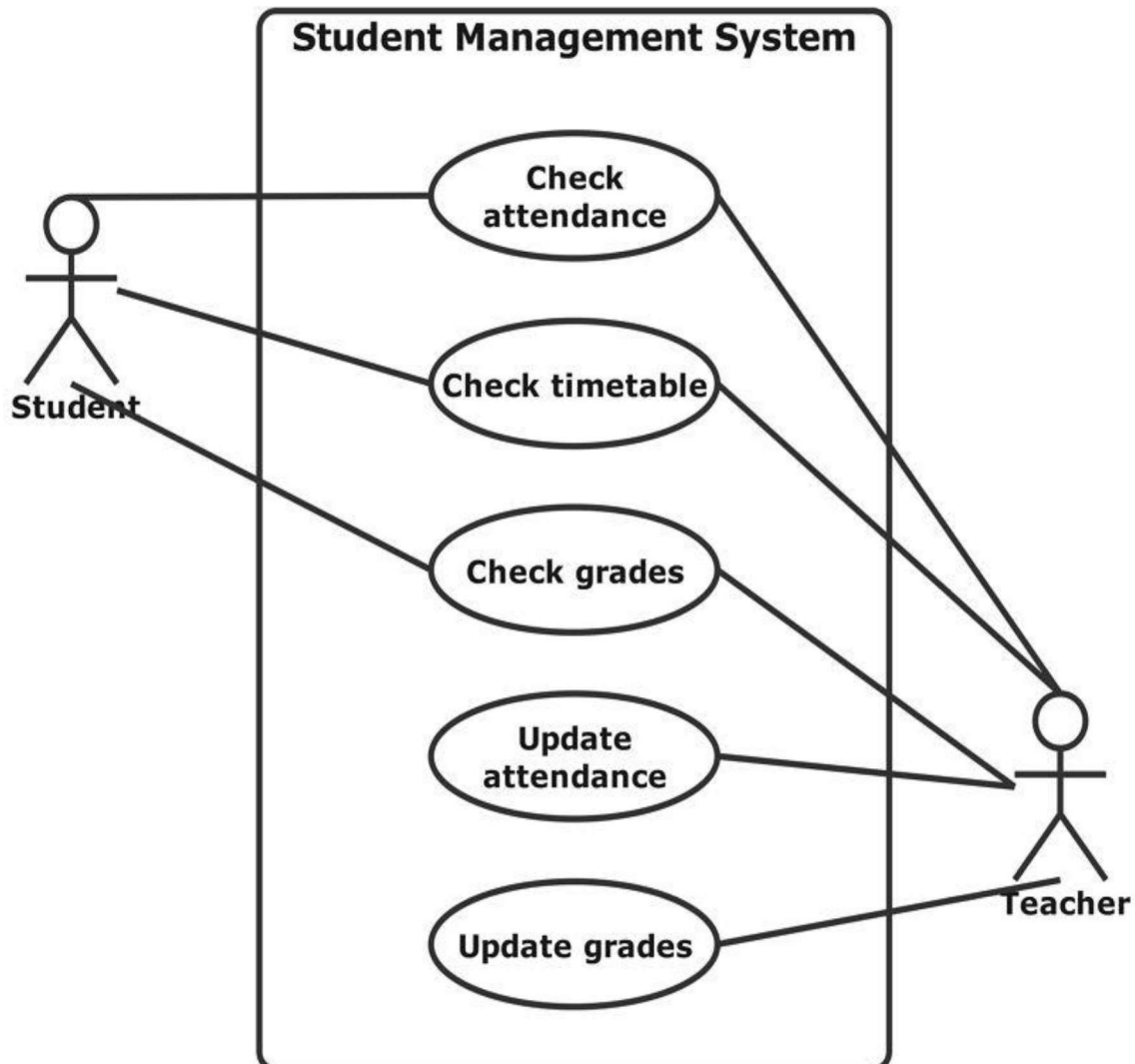


Use Case Diagram Sample

Here are the sample of UML use case diagram:

Sample 1: Online shopping system Use Case diagram



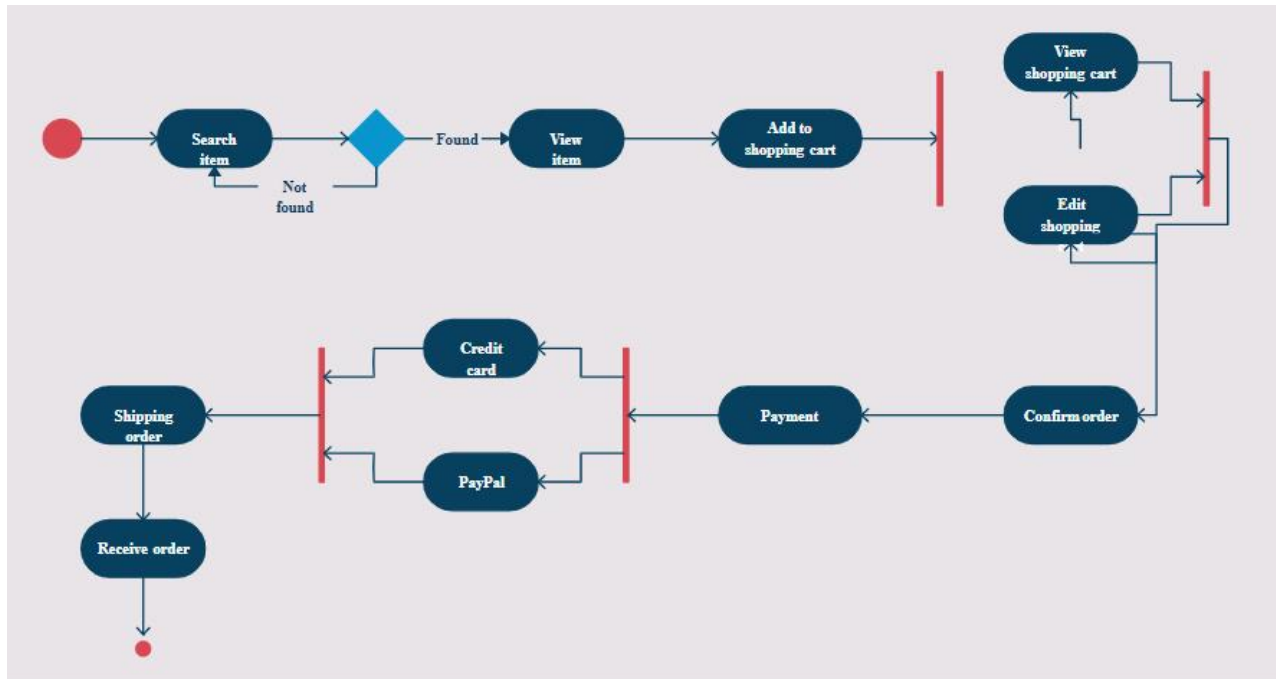


Sample 2: Student Management System Use Case Diagram



EXPERIMENT – 3

AIM: Draw the activity diagram for a given problem.





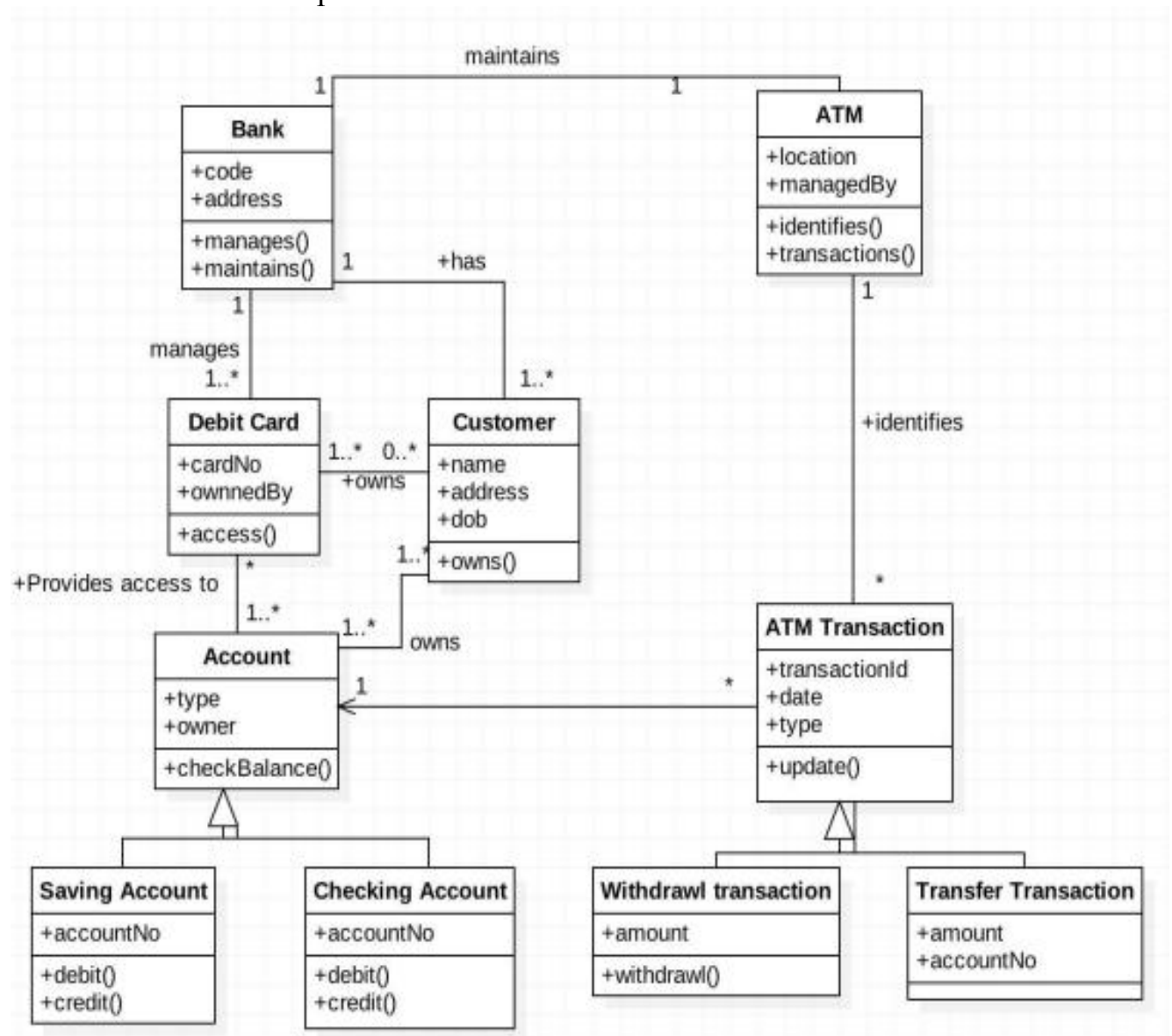
EXPERIMENT - 4

AIM: Identify the classes. Classify them as weak and strong classes and draw the class diagram for a given problem.

CLASS DIAGRAM: Class diagrams describe the static structure of a system, or how it is structured rather than how it behaves.

These diagrams contain the following elements:

1. **Classes**, which represent entities with common characteristics or features. These features include attributes, operations, and associations.
2. **Associations**, which represent relationships that relate two or more other classes where the relationships have common characteristics or features. These features include attributes and operations.





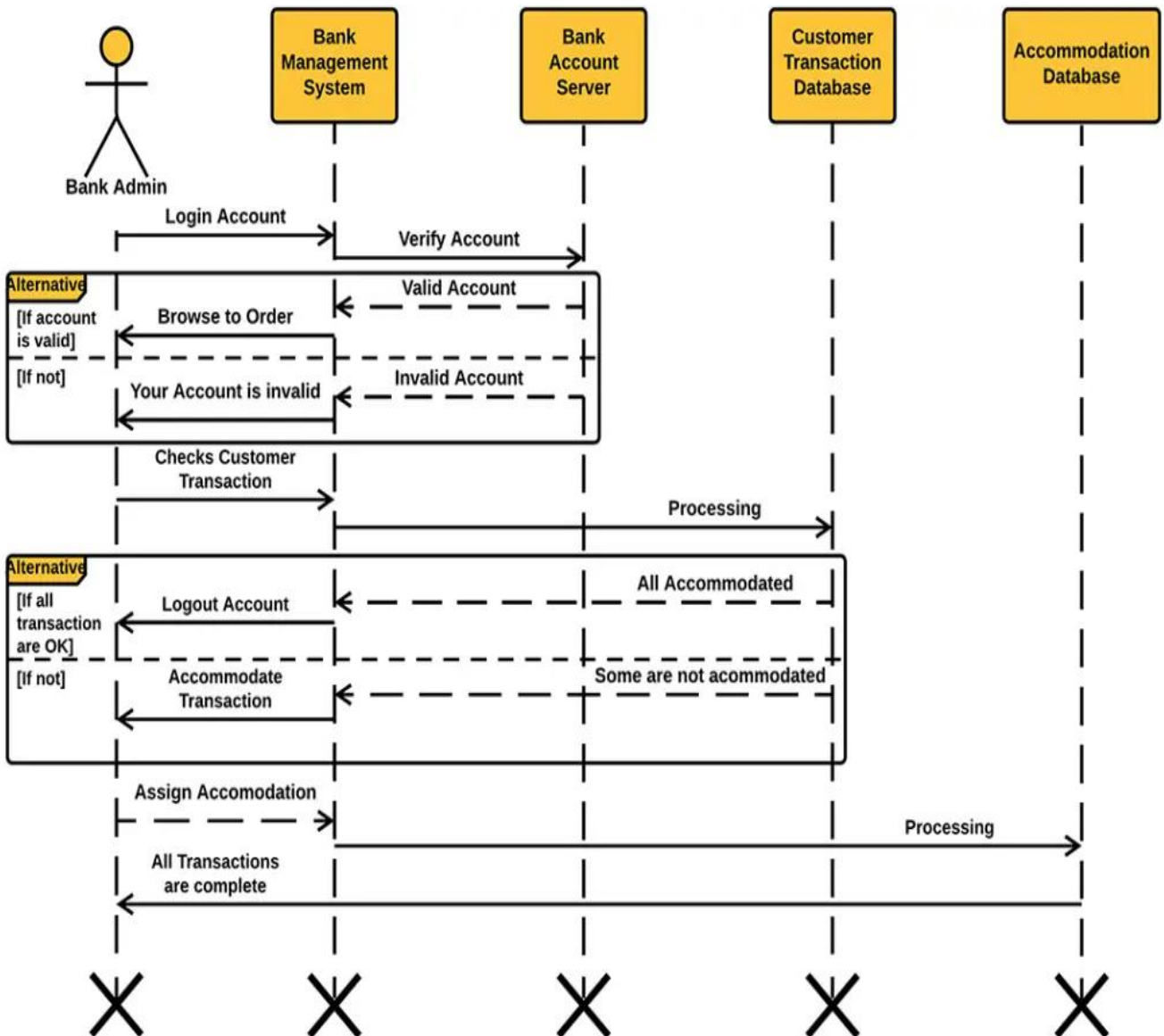
EXPERIMENT – 5

AIM: Draw the sequence diagram for These two scenarios for a given problem

THEORY: A sequence diagram is a type of UML (Unified Modeling Language) diagram that shows interactions between objects or components in a system over time. It is used in software engineering to visualize the flow of messages between objects, and is particularly useful for modeling complex systems with many interacting components.

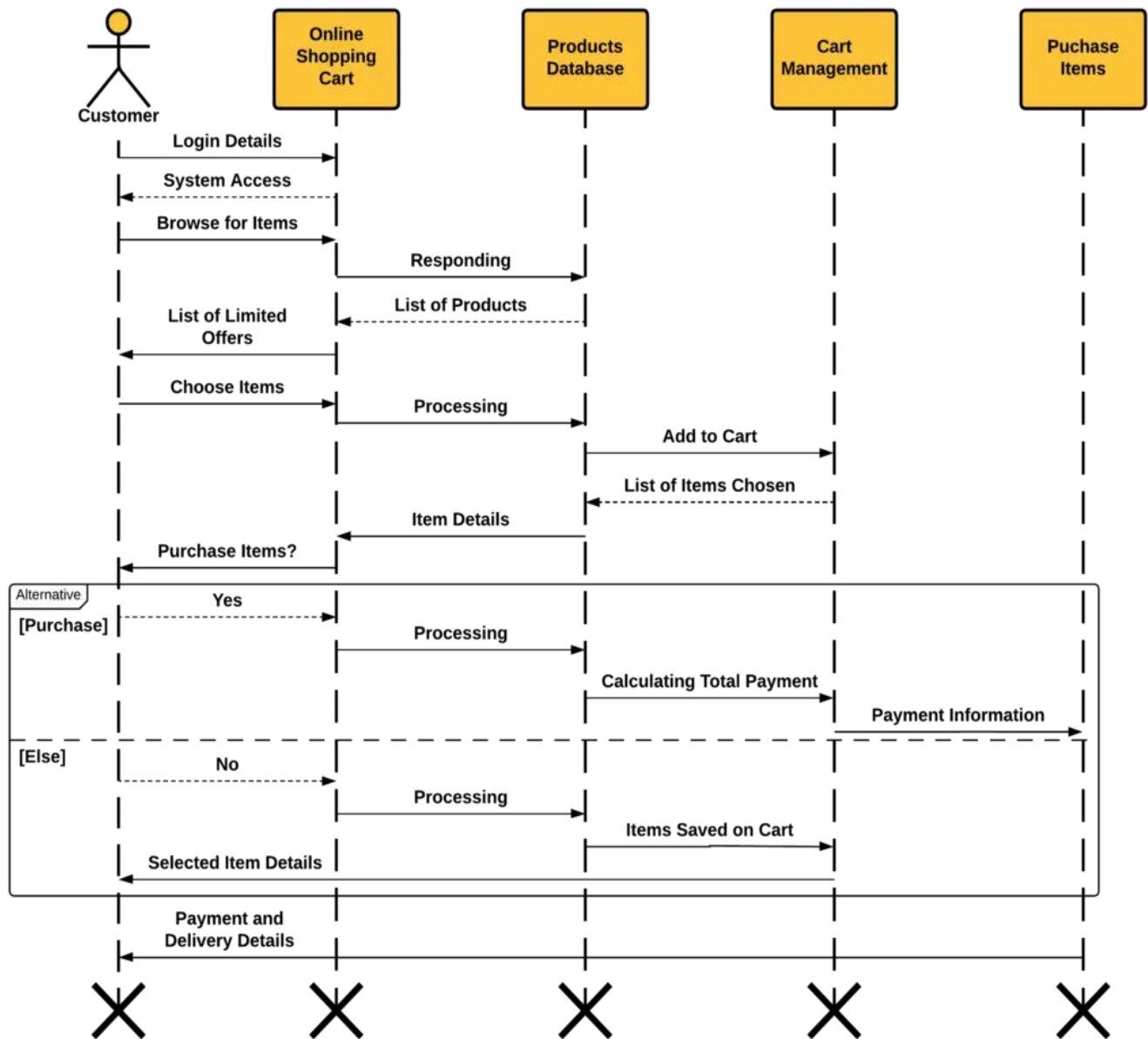
DIAGRAM:

1). BANK MANAGEMENT SYSTEM -





2). ONLINE SHOPPING SYSTEM :-



CONCLUSION:

- The sequence diagram is drawn for the given problem.

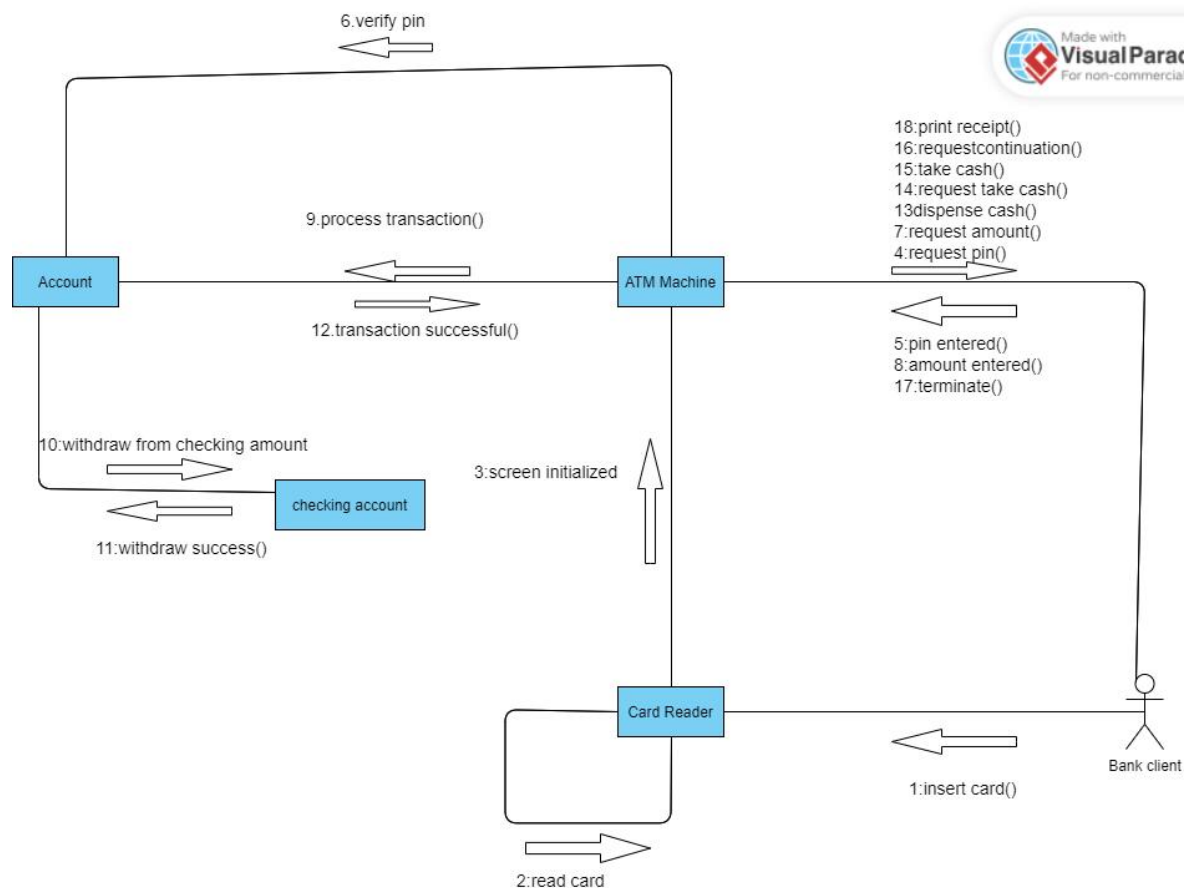


EXPERIMENT – 6

AIM: Draw the collaboration diagram for online shopping system.

THEORY: A collaboration diagram, also known as a communication diagram, is a type of interaction diagram that shows the interactions and communications between objects or roles in a system. It is a visual representation of how objects or roles collaborate with each other to achieve a specific task or goal.

DIAGRAM:



CONCLUSION:

- The collaboration diagram is drawn for the given problem.

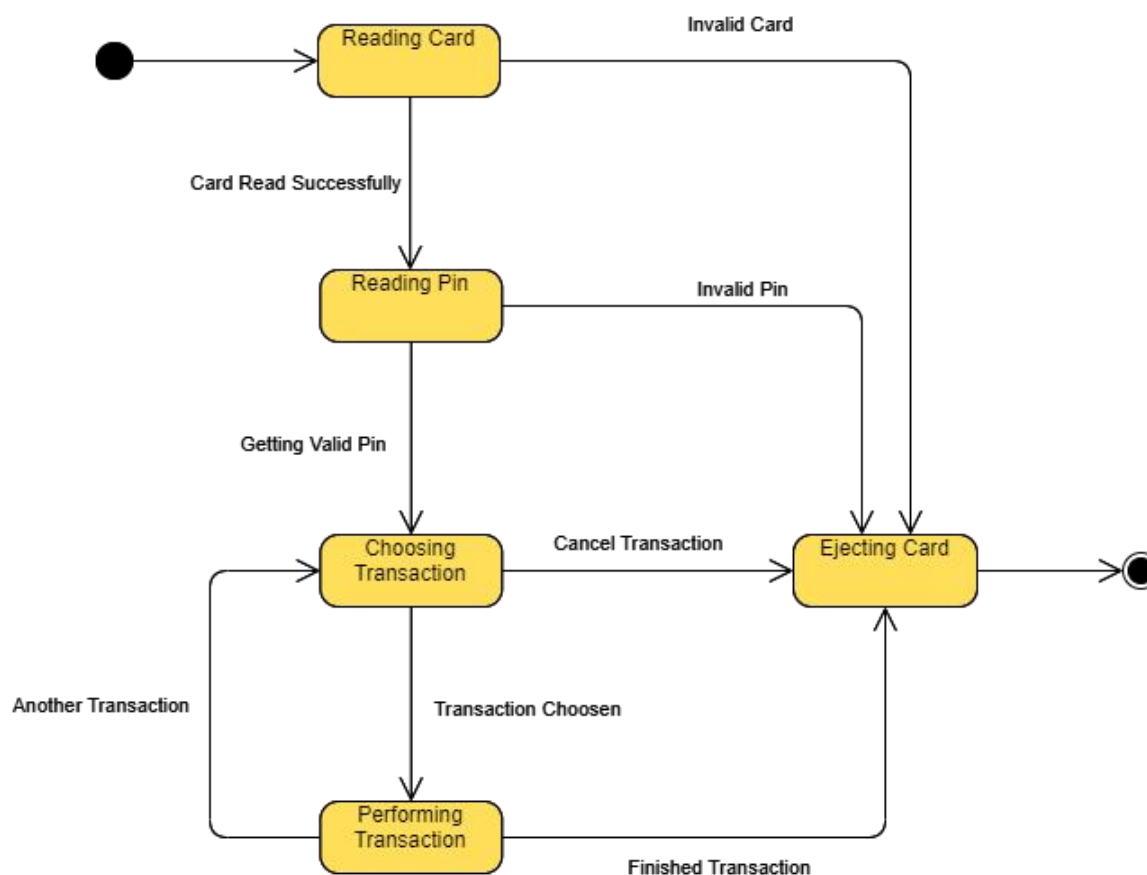


EXPERIMENT – 7

AIM: Draw the state chart diagram for ATM Machine Transaction Processing.

THEORY: A chart diagram is a graphical representation of a software system or a specific aspect of it. It is a visual representation of software components, their relationships, and interactions. The purpose of a chart diagram is to help software developers understand the structure of a software system, its components, and their interdependencies.

DIAGRAM:



CONCLUSION:

- The state chart diagram for drawn ATM Machine Transaction Processing.

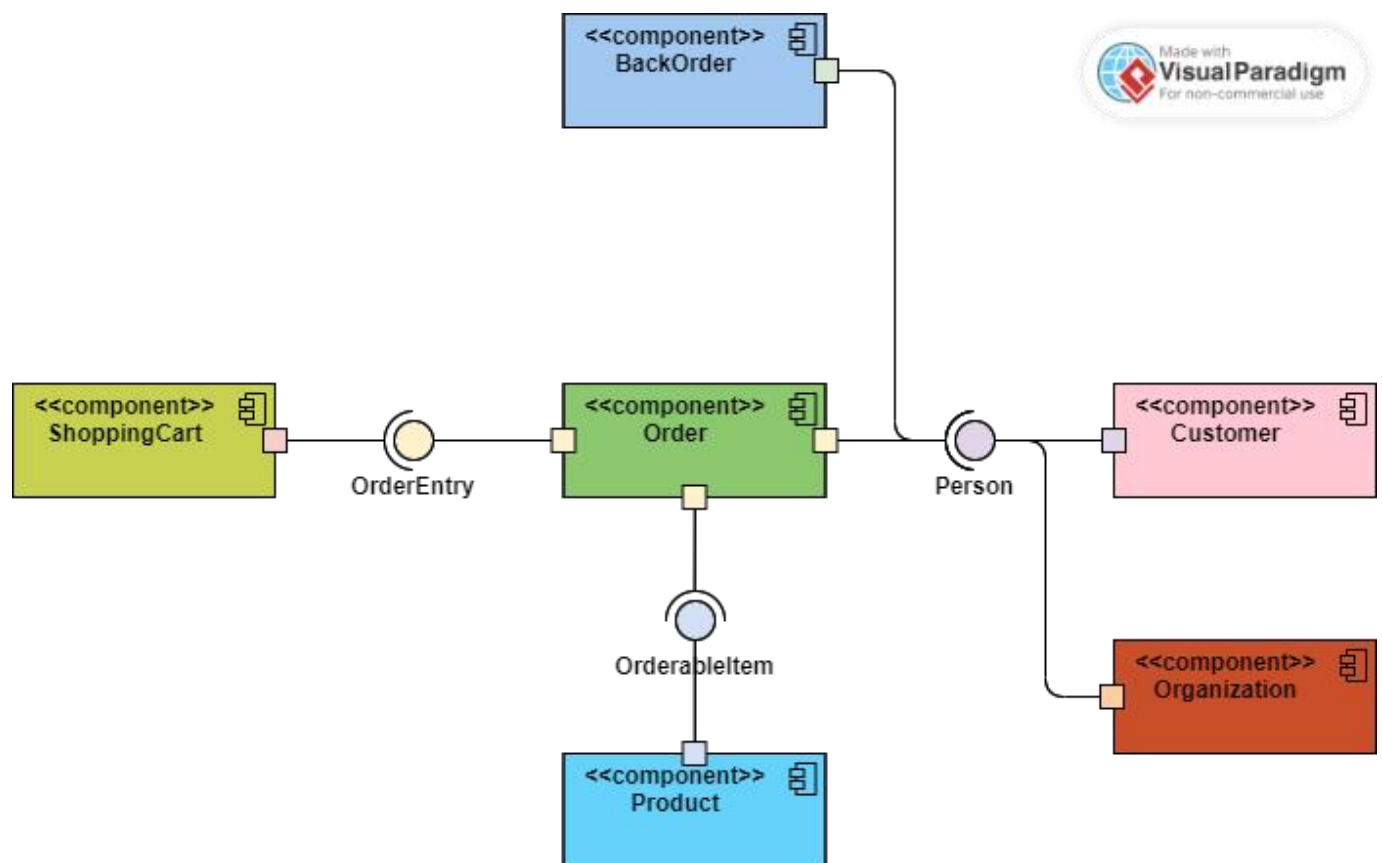


EXPERIMENT – 8

AIM: Draw the component diagram for online shopping.

THEORY: A component diagram can be used to represent the overall architecture of a software system, including the different components and how they interact with each other. It can also be used to illustrate the relationships between components and their interfaces, dependencies, and associations.

DIAGRAM:



Conclusion :-

- The component diagram drawn for online shopping.

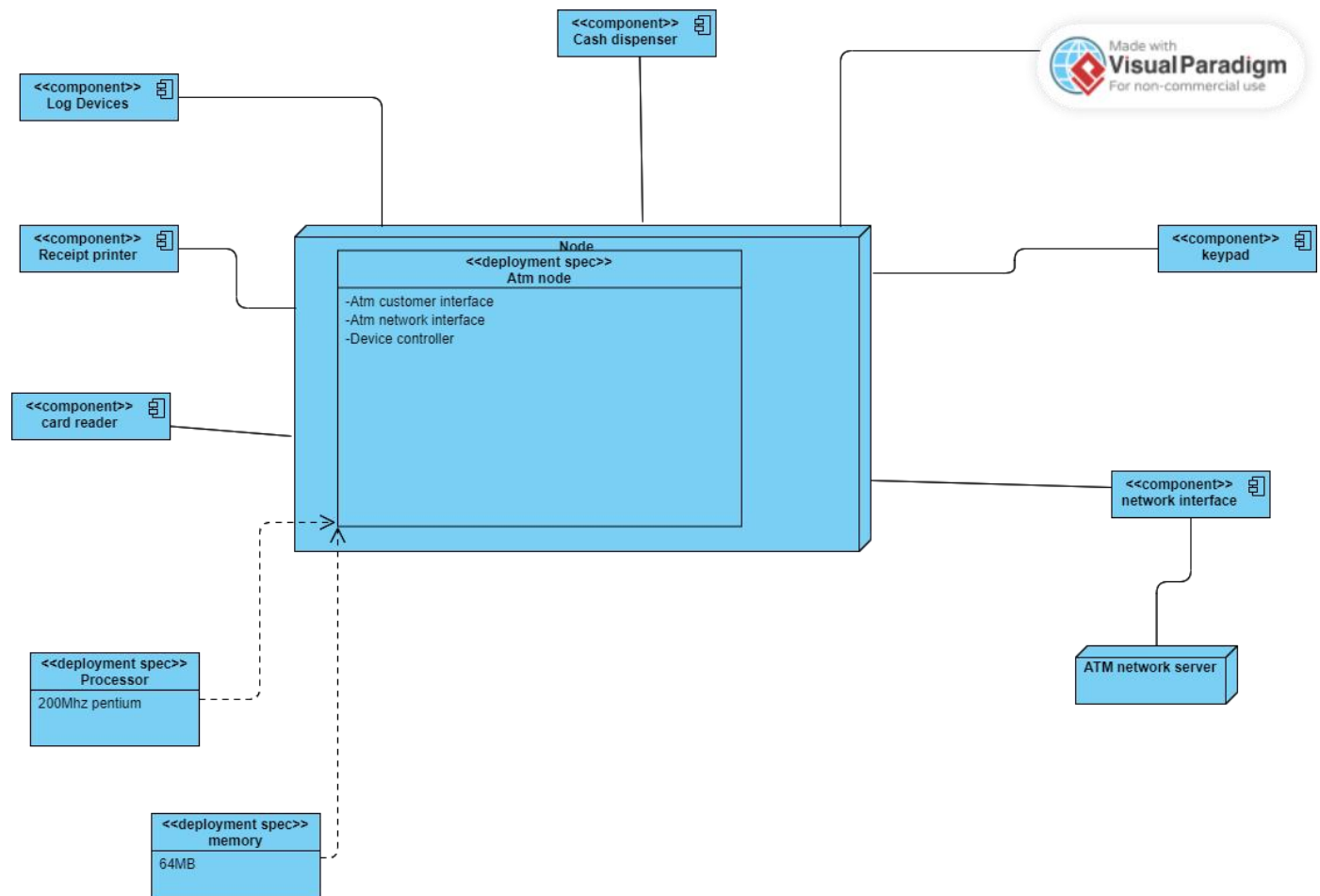


EXPERIMENT – 9

AIM: Draw the deployment diagram for Hospital Management System.

THEORY: A deployment diagram consists of nodes, which are physical entities, and components, which are software entities. Nodes can represent devices such as servers, computers, or mobile devices, while components can represent software applications, modules, or libraries. The connections between nodes and components represent communication channels or network protocols used for inter-component communication.

DIAGRAM:



CONCLUSION:

- The deployment diagram drawn for Hospital Management System.



EXPERIMENT – 10

AIM:- Draw Data Flow Diagram for Bank Account.

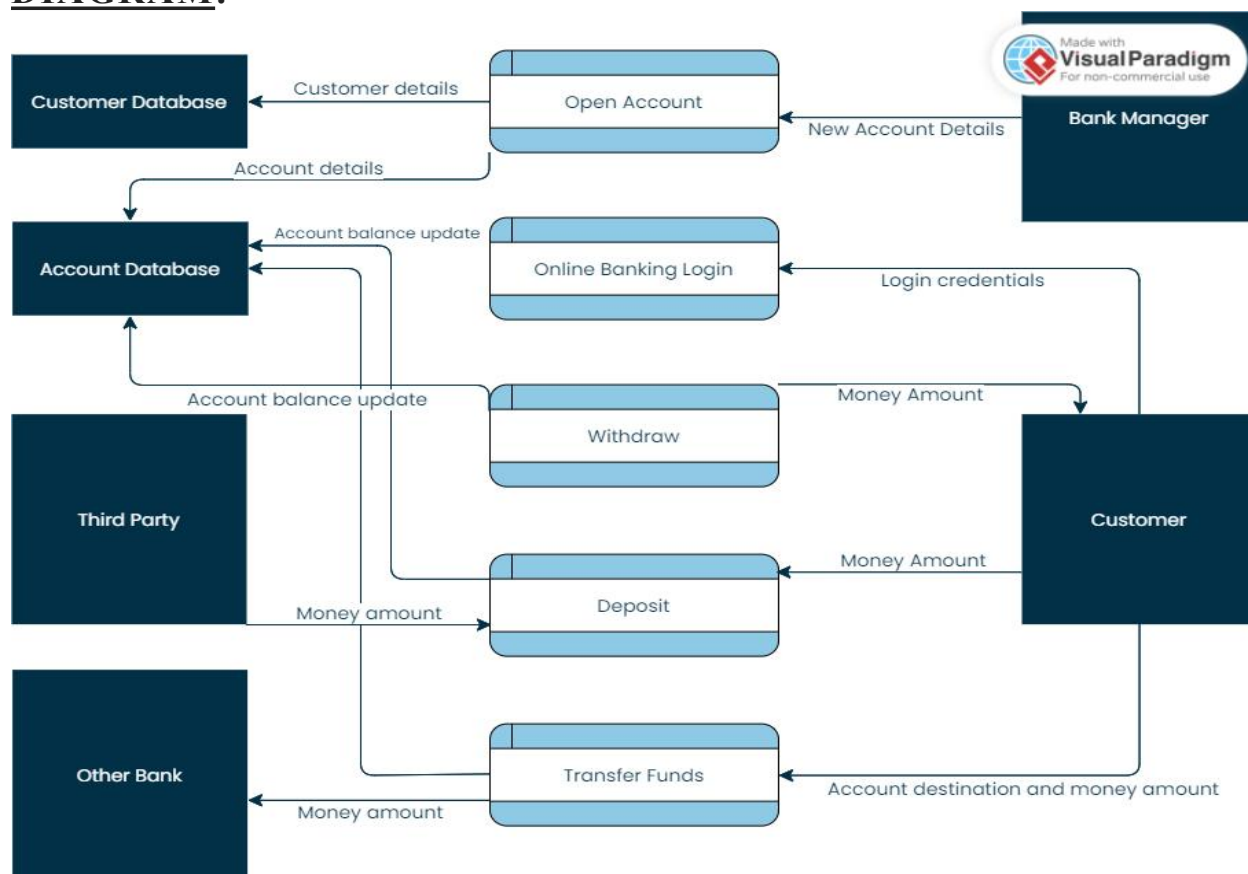
THEORY:- A data flow diagram (DFD) is a graphical representation of the flow of data through a system. It is used to show how data moves between different processes, entities, and data stores within a system.

To draw a data flow diagram, you first need to identify the inputs, outputs, processes, and data stores in the system. You can then use symbols to represent these components and show how data flows between them.

There are four main components in a DFD:

1. Entities: These are the external entities that send or receive data to or from the system.
2. Processes: These are the operations or transformations that are performed on the data.
Data
3. Flows: These represent the movement of data between the entities, processes, and data
4. stores. Data Stores: These are the places where data is stored within the system.

DIAGRAM:





EXPERIMENT – 11

AIM: Write a program to compute cyclomatic complexity.

THEORY: Cyclomatic complexity is a software metric that measures the complexity of a program based on the number of independent paths through its source code. The higher the cyclomatic complexity, the more complex the program is considered to be, which can make it more difficult to understand, test, and maintain.

CODE:

```
#include <stdio.h>

int main() {
    int nodes, edges, connected_components;

    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    printf("Enter the number of connected components: ");
    scanf("%d", &connected_components);

    int cyclomatic_complexity = edges - nodes + 2 * connected_components;

    printf("Cyclomatic Complexity: %d\n", cyclomatic_complexity);

    return 0;
}
```

OUTPUT:

```
Output
/tmp/GP6yekxP7T.o
Enter the number of nodes: 5
Enter the number of edges: 5
Enter the number of connected components: 3
Cyclomatic Complexity: 6
```

CONCLUSION:

- The cyclomatic complexity is calculated for the given problem.



EXPERIMENT – 12

AIM: Write a program to compute Halstead's Complexity.

THEORY: Halstead complexity measures are software metrics introduced by Maurice Howard Halstead in 1977 as part of his treatise on establishing an empirical science of software development. Halstead makes the observation that metrics of the software should reflect the implementation or expression of algorithms in different languages, but be independent of their execution on a specific platform. These metrics are therefore computed statically from the code.

CODE:

```
#include <stdio.h>
#include <math.h>
int main()
{
    float op_distinct, opd_distinct, op_total, opd_total;
    float length, vocab_size, volume, level, difficulty, effort;
    float time;
    float k = 18;
    printf("\n Enter the value of number of distinct operators: ");
    scanf("%f", &op_distinct);
    printf("\n Enter the value of number of distinct operands: ");
    scanf("%f", &opd_distinct);
    printf("\n Enter the value of total number of operators: ");
    scanf("%f", &op_total);
    printf("\n Enter the value of total number of operands: ");
    scanf("%f", &opd_total);
    vocab_size = op_distinct + opd_distinct;
    length = op_total + opd_total;
    volume = length * log(vocab_size);
    level = (op_distinct / 2) * (opd_total / opd_distinct);
    difficulty = 1 / level;
    effort = volume * difficulty;
```



```
time = effort / k;  
printf("\n Program Vocabulary: %f", vocab_size);  
  
printf("\n Program Length: %f", length);  
printf("\n Calculated Program Length: %f", volume);  
printf("\n Volume: %f", volume);  
printf("\n Difficulty: %f", difficulty);  
printf("\n Effort: %f", effort);  
printf("\n Time to implement: %f", time);  
return 0;  
}
```

OUTPUT:

Output

```
/tmp/6qV0bu8FWN.o  
Enter the value of number of distinct operators: 3  
Enter the value of number of distinct operands: 4  
Enter the value of total number of operators: 8  
Enter the value of total number of operands: 12  
Program Vocabulary: 7.000000  
Program Length: 20.000000  
Calculated Program Length: 38.918201  
Volume: 38.918201  
Difficulty: 0.222222  
Effort: 8.648489  
Time to implement: 0.480472
```

CONCLUSION:

- The Halstead's Complexity is calculated for the given problem.