MINI PROJECT REPORT
On
TODO LIST APPLICATION USING JAVA

Submitted for partial fulfillment of the award of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE & ENGINEERING
(2025–2026)

AMBALIKA INSTITUTE OF MANAGEMENT & TECHNOLOGY, LUCKNOW
Affiliated to
Dr. A.P.J. Abdul Kalam Technical University, Lucknow

Submitted To: Ms. KM DIVYA

_____

(Project Guide)
Department of
CSE

Submitted By:

Name: Mohd Ayan
Roll Number: 2203630100091
Name: Noor Alam
Roll Number:2203630100102
Name: Deepsikha
Roll Number:2203620100057
Name: Krish Mohan Sharma
Roll Number: 2203630100080

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project entitled **"Todo List Application Using Java"** submitted by **Mohd Ayan , Noor Alam, Deepsikha Singh, Krish Mohan Sharma** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering** from **Dr. A.P.J. Abdul Kalam Technical University, Lucknow**, is a record of his own work carried out under my supervision and guidance.

The project report embodies the results of original work and the contents of this project have not been submitted to any other university or institute for the award of any degree.

Signature of Guide

Head of Department(CSE)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

I hereby declare that the mini project titled **"Todo List Application Using Java"** submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering** from **Dr. A.P.J. Abdul Kalam Technical University, Lucknow**, is my original work.

This project has not been submitted to any other university or institution for the award of any degree or diploma.

Name: Mohd Ayan
Roll Number: 2203630100091
Name: Noor Alam
Roll Number: 2203630100102
Name: Deepsikha Singh
Roll Number: 2203630100057
Name: Krish Mohan Sharma
Roll Number: 2203630100080

Date: _____

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project guide **Ms. KM. Divya** for her continuous guidance, encouragement, and valuable suggestions throughout the development of this mini project.

I am also thankful to the Head of the Department and all faculty members of the **Computer Science & Engineering Department, AIMT Lucknow**, for providing the necessary facilities and support.

Finally, I would like to thank my friends and family members for their constant motivation and support during the successful completion of this project.

Name: Mohd Ayan
Roll Number: 2203630100091
Name: Noor Alam
Roll Number: 2203630100102
Name: Deepsikha Singh
Roll Number: 2203630100057
Name: Krish Mohan Sharma
Roll Number: 2203630100080

PREFACE

This mini project, **"Todo List Application Using Java"**, has been developed as part of the academic curriculum to gain practical knowledge of Java programming and software development concepts.

The project focuses on creating a simple yet effective task management system that helps users organize their daily activities efficiently. Through this project, I gained hands-on experience in Java, object-oriented programming concepts, and basic application design.

This report provides detailed information about the project objectives, tools used, system design, implementation, and results.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Table of Contents

ABSTRACT

In the modern digital era, effective task management plays a crucial role in improving productivity and time management. A **Todo List Application** is a simple yet powerful tool that helps users organize, track, and manage their daily tasks efficiently.

This mini project titled **"Todo List Application Using Java"** focuses on developing a console-based/GUI-based Java application that allows users to add, update, delete, and view tasks. The application is designed using object-oriented programming principles to ensure modularity, reusability, and scalability.

The project provides hands-on experience with Java fundamentals such as classes, objects, methods, collections, and file handling. It also helps in understanding real-world application development and problem-solving using Java programming language.

INTRODUCTION

Purpose of the Project

The primary purpose of the To-Do List Application is to provide an effective and reliable solution for managing daily tasks digitally. In the absence of a structured task management system, users often rely on memory or manual notes, which can lead to inefficiency and missed responsibilities. This application aims to overcome these limitations by offering a centralized platform where tasks can be recorded, tracked, and updated easily.

The project also serves an educational purpose by helping students understand real-world software development practices such as requirement analysis, system design, coding, and testing.

Project Overview

The To-Do List Application is a standalone software system developed to assist users in managing tasks efficiently. The system provides a graphical interface through which users can add new tasks, view existing tasks, mark tasks as completed, and delete tasks when they are no longer required. The application ensures that task data is saved locally, allowing users to retrieve their task list even after restarting the application.

Project Scope

The scope of this project is limited to a single-user task management system. The application focuses on basic task operations such as task creation, deletion, and status updates. It does not include advanced features such as cloud storage, user authentication, or multi-device synchronization. However, the current scope provides a strong foundation for future enhancements.

Overview of the Project

This project is designed using a modular approach, where different functionalities are divided into independent modules. The system architecture separates the user interface, application logic, and data storage components, ensuring better maintainability and scalability.

Problem Area Description

In daily life, individuals face challenges in managing multiple responsibilities efficiently. Traditional methods such as notebooks or mental reminders are unreliable and prone to errors. The lack of a proper task tracking mechanism often results in incomplete or forgotten tasks. The To-Do List Application addresses this problem by providing a digital solution that ensures systematic task management.

SYSTEM ANALYSIS

System analysis is a crucial phase in software development where the existing system is studied, and the requirements of the proposed system are analyzed in detail. This phase helps in understanding the problem domain and designing an efficient solution.

Existing System

In the existing system, task management is generally carried out using traditional manual methods such as notebooks, diaries, or memory-based planning. These methods have been in use for a long time but suffer from several limitations.

Limitations of the Existing System

1. **Manual Task Tracking**
   Tasks are written manually, which requires extra effort and time. Maintaining a physical notebook is inconvenient and not always accessible
2. **No Task Status Monitoring**
   The existing system does not provide a mechanism to track whether a task is completed, pending, or overdue.
3. **High Probability of Data Loss**
   Paper-based records can be easily lost, damaged, or misplaced, leading to loss of important information.
4. **Lack of Organization**
   Managing a large number of tasks becomes difficult due to poor organization and lack of categorization.
5. **Inefficiency and Time Consumption**
   Searching for a specific task in a notebook is time-consuming and inefficient.

Proposed System

The proposed system is a **digital To-Do List Application** that overcomes the drawbacks of the existing system. It provides a modern, efficient, and reliable solution for task management.

Features of the Proposed System

1. **Digital Task Management**
   Users can add tasks digitally through a graphical interface, eliminating the need for manual writing.

2. **Task Completion Tracking**
   Tasks can be marked as completed, allowing users to track progress easily.
3. **Persistent Storage**
   The system saves tasks locally, ensuring data is not lost even after the application is closed.
4. **User-Friendly Interface**
   The graphical interface is simple, intuitive, and easy to use.
5. **Improved Efficiency**
   The system reduces time and effort required to manage tasks.
6. **Error Reduction**
   Digital storage minimizes the chances of losing task data

Comparison Between Existing and Proposed System

Feature	Existing System	Proposed System
Task Management	Manual	Digital
Data Storage	Paper-based	File-based
Task Tracking	Not available	Available
Data Security	Low	High
Efficiency	Low	High
User Interface	Not applicable	Graphical UI

Overview of System Analysis

System analysis helps in identifying system requirements, understanding user needs, and defining system constraints. In this project, system analysis ensured that the application meets user expectations while remaining simple and efficient.

FEASIBILITY REPORT

Feasibility analysis determines whether the proposed system is viable from different perspectives. The feasibility study ensures that the system can be developed and implemented successfully within available resources.

Operational Feasibility

Operational feasibility refers to the ease with which the system can be operated by users.

Operational Advantages

1. **Ease of Use**
   The application requires no technical expertise to operate. Users can interact with the system using simple buttons and text fields.
2. **Minimal Training Required**
   Since the interface is intuitive, users do not require any training.
3. **User Acceptance**
   The system is likely to be accepted by users due to its simplicity and usefulness.
4. **Improved Productivity**
   By organizing tasks digitally, users can manage time more effectively.

Technical Feasibility

Technical feasibility analyzes whether the required technology is available and suitable.

Technical Considerations

1. **Technology Availability**
   The system uses Java and GUI libraries, which are widely available and well-supported.
2. **System Compatibility**
   The application can run on standard operating systems such as Windows and Linux.
3. **Scalability**
   The system can be enhanced in the future to support additional features.
4. **Maintainability**
   Modular design ensures easy maintenance and updates.

Financial & Economical Feasibility

Financial feasibility evaluates the cost-effectiveness of the project.

Cost Analysis

1. **Software Cost**
   The application uses open-source software and tools, resulting in zero software cost.
2. **Hardware Cost**
   No additional hardware is required beyond a standard personal computer.
3. **Development Cost**
   Development cost is minimal as it is a student mini project.
4. **Economic Benefit**
   The system saves time and effort, increasing overall productivity.

Feasibility Conclusion

After analyzing operational, technical, and financial aspects, it is concluded that the To-Do List Application is **highly feasible** and suitable for implementation.

SYSTEM REQUIREMENT SPECIFICATION (SRS)

The System Requirement Specification (SRS) document describes the complete behavior, functionality, and constraints of the To-Do List Application. It serves as a formal agreement between the developer and the user regarding system requirements and ensures that the final system meets user expectations.

Introduction to SRS

The SRS provides a detailed description of the functional and non-functional requirements of the system. It defines what the system will do, how it will perform, and the limitations under which it will operate. This document is essential for designing, implementing, and testing the system efficiently.

Purpose of the SRS Document

The main objectives of this SRS are:

- To define system functionality clearly
- To identify system constraints and assumptions
- To act as a reference for developers and testers
- To ensure user satisfaction

Overall Description

Product Perspective

The To-Do List Application is a standalone system designed for individual users. It does not depend on any external system or network service. The application interacts directly with the user through a graphical interface and stores data locally.

Product Functions

The primary functions of the system include:

- Creating tasks
- Displaying tasks
- Updating task status
- Deleting tasks
- Saving and retrieving task data

User Characteristics

The intended users of the system include:

- Students
- Professionals
- Home users

Users are expected to have basic computer knowledge but no technical expertise.

Operating Environment

- Operating System: Windows / Linux
- Programming Language: Java
- IDE: IntelliJ, Eclipse, or VS Code
- Storage: Local file system

Design and Implementation Constraints

- Local storage only
- Single-user access
- No internet connectivity
- Limited system resources

Functional Requirements

Functional requirements describe the specific behavior and functions of the system.

Add Task

- The system shall allow the user to add a new task.
- The system shall validate input to prevent empty tasks.
- The task shall be added to the task list immediately.

View Task

- The system shall display all existing tasks.
- Completed tasks shall be visually distinguishable.

Update Task Status

- The system shall allow the user to mark a task as completed.

- Completed tasks shall be indicated clearly.

Delete Task

- The system shall allow the user to delete selected tasks.
- Deleted tasks shall be removed permanently.

Save Task Data

- The system shall save all tasks automatically.
- Data shall persist after application restart.

Load Task Data

- The system shall load saved tasks at startup.
- The system shall handle missing or empty files safely.

Non-Functional Requirements

Non-functional requirements define system quality attributes.

Usability

- The interface shall be simple and intuitive.
- Users shall be able to perform tasks easily.

Performance

- The system shall respond quickly to user actions.
- Task operations shall execute without noticeable delay.

Reliability

- The system shall store data safely.
- The application shall not crash during normal use.

Security

- Data shall be stored locally and securely.
- Unauthorized access is minimized due to single-user design.

Maintainability

- The system shall be easy to modify and update.
- Code shall follow modular design principles.

Portability

- The system shall run on multiple operating systems.
- Platform dependency shall be minimal.

System Components

The To-Do List Application consists of the following components:

User Interface Component

- Handles user input
- Displays tasks
- Provides buttons and controls

Task Management Component

- Manages task operations
- Updates task status
- Handles deletions

Storage Component

- Saves task data
- Loads task data
- Manages file handling

Control Component

- Coordinates interaction between modules
- Controls application flow

System Interaction

The interaction between system components follows a structured flow:

1. User interacts with UI
2. UI sends request to task manager
3. Task manager processes request
4. Storage module saves or retrieves data
5. UI updates output

Constraints

The system operates under the following constraints:

- Single user access
- No database integration

- Local file-based storage
- Limited hardware resources

User Roles

User

- Add tasks
- View tasks
- Mark tasks as completed
- Delete tasks

(Note: No admin role is present in this system.)

Module Description

Input Module

Accepts task input from the user and validates it.

Display Module

Displays task list and task status.

Storage Module

Handles saving and loading of task data.

Control Module

Manages communication between all modules.

SDLC METHODOLOGIES

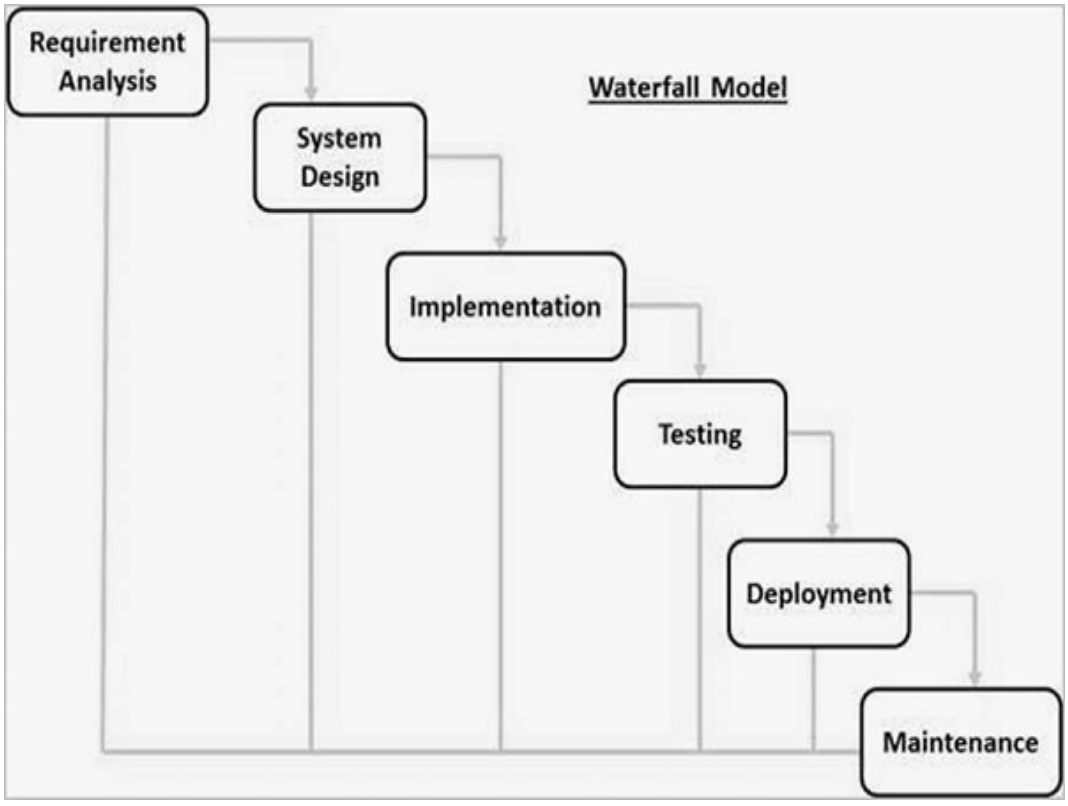
Software Development Life Cycle (SDLC) defines the process followed to design, develop, test, and deploy a software system. In this project, a systematic SDLC approach has been adopted to ensure the quality, reliability, and maintainability of the To-Do List Application.

Introduction to SDLC

The Software Development Life Cycle consists of a series of well-defined phases that guide developers from the initial concept to final deployment. Each phase produces specific deliverables and ensures proper planning and execution of the project.

SDLC Model Used – Waterfall Model

The **Waterfall Model** is chosen for this project because of its simplicity and structured approach. It is suitable for small-scale projects with clearly defined requirements.



Phases of the Waterfall Model

Requirement Analysis

In this phase, the system requirements are identified and analyzed. User needs are studied, and functional as well as non-functional requirements are documented.

System Design

The system architecture is designed based on the requirements. This includes GUI layout, data storage design, and module interaction.

Implementation

In this phase, the actual coding of the application is carried out using Java programming language. Each module is developed independently and later integrated.

Testing

Testing ensures that the system works as expected. Various test cases are applied to check functionality, usability, and reliability.

Deployment

After successful testing, the application is deployed on the user's system.

Maintenance

Maintenance includes fixing bugs, improving performance, and adding new features in the future.

Advantages of Using Waterfall Model

- Simple and easy to understand
- Clear documentation

SOFTWARE REQUIREMENTS

S.No	Software Component	Specification / Description
1.	Operating System	Windows 10 / Windows 11 / Linux
2.	Programming Language	Java (JDK 8 or above)
3.	IDE / Editor	IntelliJ IDEA / Eclipse / VS Code
4.	GUI Framework	Java Swing
5.	Database	File-based storage (Text File)
6.	Libraries Used	Java Standard Libraries
7.	Build Tool	Java Compiler (javac)
8.	Version Control (Optional)	Git
9.	Documentation Tool	MS Word / Google Docs
10.	PDF Reader	Adobe Reader

HARDWARE REQUIREMENTS

S.No	Hardware Component	Minimum Specification
1	Processor	Intel i3 or equivalent
2	RAM	4GB
3	Hard Disk	10 GB free space
4	Input Devices	Keyboard, Mouse
5	Output Devices	Monitor
6	System Type	Desktop / Laptop
7	Architecture	64-bit System
8	Network	NotRequired
9	Display Resolution	1024 × 768 or higher

SYSTEM DESIGN

System design describes how the system components are organized and interact with each other. A well-designed system improves performance, scalability, and maintainability.

Architectural Design

The To-Do List Application follows a **layered architecture**, consisting of:

1. Presentation Layer
2. Application Logic Layer
3. Data Storage Layer

Presentation Layer (GUI Design)

This layer interacts directly with the user. It includes:

- Input text field for tasks
- Buttons for add and delete
- Task display list

The interface is designed to be simple, clean, and user-friendly.

Application Logic Layer

This layer handles:

- Task creation
- Task deletion
- Task status update
- Input validation

It ensures smooth communication between the GUI and storage modules.

Data Storage Layer

This layer handles:

- Saving task data
- Loading task data
- Managing file operations

The system uses file-based storage for simplicity and efficiency.

Advantages of System Design

- Easy to understand
- Modular structure
- Easy maintenance

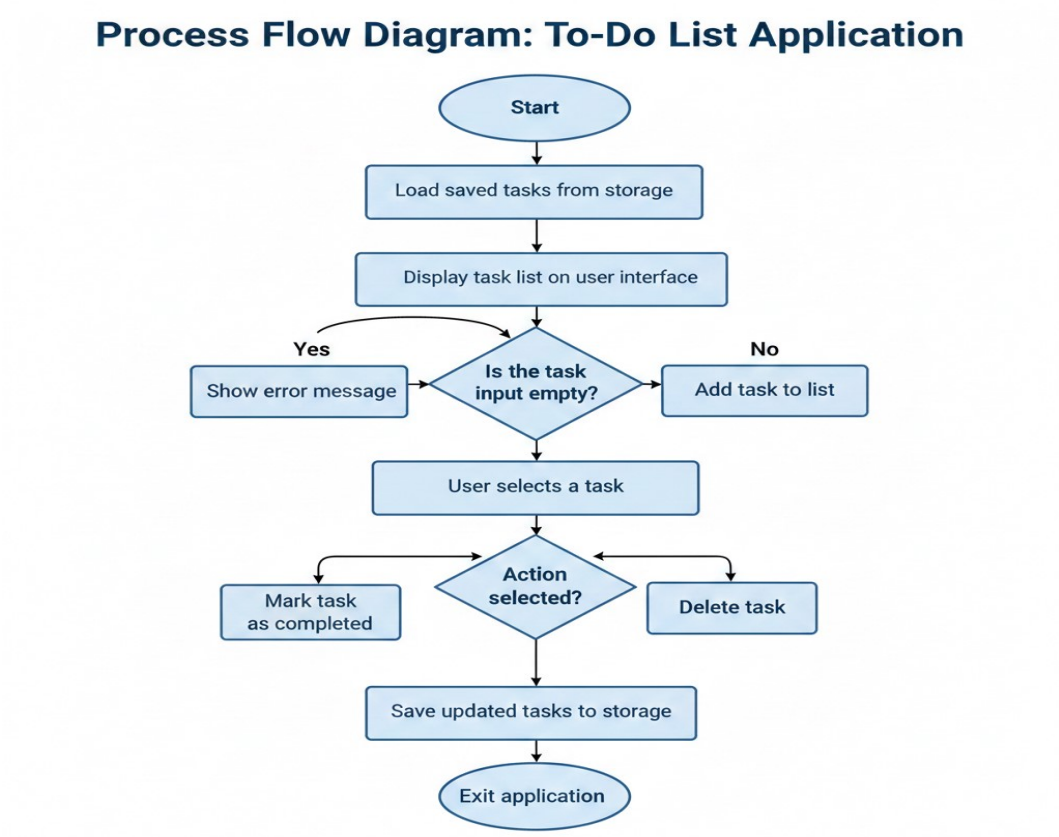
PROCESS FLOW

Process flow represents the sequence of steps involved in the execution of the To-Do List Application. It explains how the system behaves from the moment it starts until it is closed.

Process Flow Description

1. The application starts.
2. Previously saved tasks are loaded from storage.
3. The user interacts with the system through the GUI.
4. The user can add a new task.
5. The user can mark tasks as completed.
6. The user can delete tasks.
7. All changes are saved automatically.
8. The application exits safely.

This flow ensures smooth interaction and data consistency.



Data Flow Diagram (DFD)

A **Data Flow Diagram (DFD)** is a graphical representation that shows how data flows within a system. It illustrates the movement of data from external entities to internal processes, data stores, and back to the external entities. DFD helps in understanding the system's functionality without focusing on technical implementation details.

DFDs are mainly used during the **system analysis and design phase** to visualize how information is processed and stored in a system.

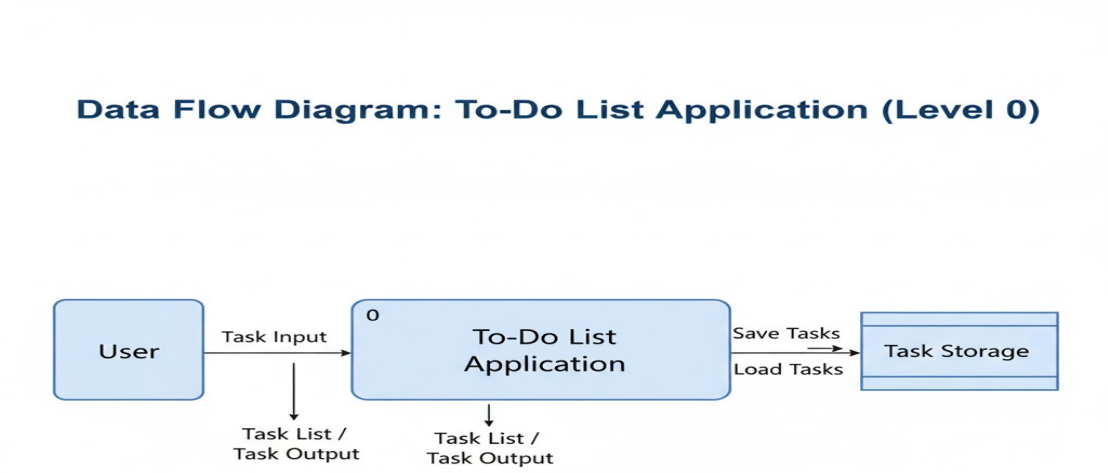
Components of a Data Flow Diagram

A DFD consists of four main components:

1. **External Entity** – Represents a user or external system that interacts with the application.
2. **Process** – Represents an activity that transforms input data into output data.
3. **Data Store** – Represents storage of data such as files or databases.
4. **Data Flow** – Represents the movement of data between entities, processes, and data stores.

DFD Level 0 (Context Diagram)

DFD Level 0, also known as the **Context Diagram**, provides a **high-level overview** of the entire system. In this level, the whole system is represented as **a single process**, showing only the interaction between the system and external entities.



Key features of DFD Level 0:

- Shows the system as one single process
- Identifies external entities interacting with the system
- Displays major data inputs and outputs
- Does not show internal process details

In the To-Do List application, DFD Level 0 shows how the user interacts with the system to add, view, and manage tasks.

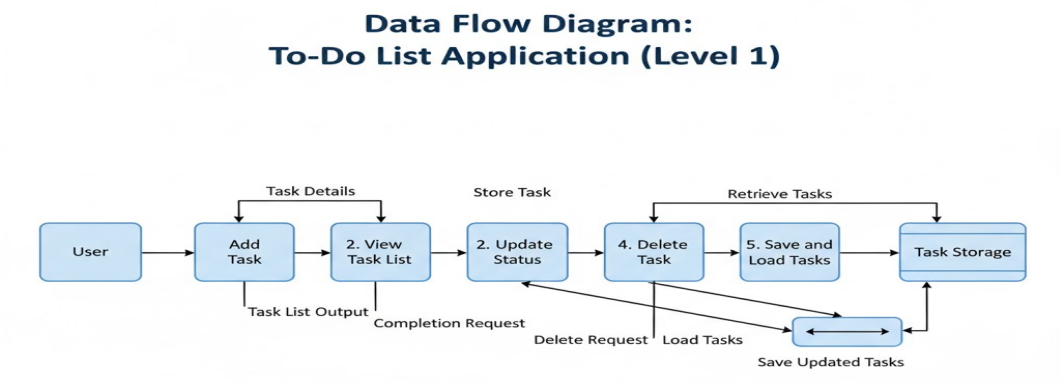
DFD Level 1

DFD Level 1 provides a **more detailed view** of the system by breaking the single process of Level 0 into **multiple sub-processes**. It explains how data flows between different internal processes and data stores.

Key features of DFD Level 1:

- Expands the main system into detailed processes
- Shows internal data flow and data storage
- Helps in understanding system functionality in detail
- Maintains consistency with Level 0 inputs and outputs

In the To-Do List application, DFD Level 1 includes processes such as adding tasks, updating task status, deleting tasks, and saving tasks to storage



Unified Modeling Language (UML)

Unified Modeling Language (UML) is a standardized visual modeling language used to design and document software systems. UML diagrams help developers and stakeholders understand the structure and behavior of a system before implementation. They provide a clear graphical representation of system components and their interactions.

UML is widely used during the **system design phase** of the Software Development Life Cycle (SDLC).

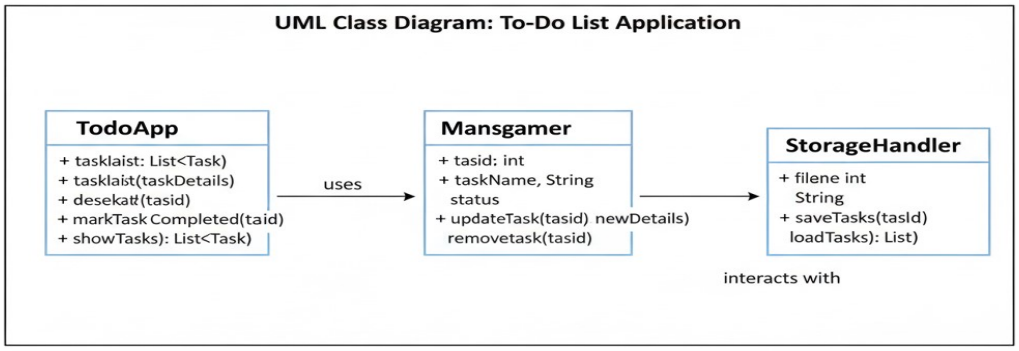
UML Class Diagram

A **Class Diagram** represents the **static structure** of a system. It shows the system's classes, their attributes, methods, and the relationships between them. Class diagrams help in understanding how the system is organized internally and support object-oriented design.

Key features of a Class Diagram:

- Shows classes with attributes and methods
- Represents relationships such as association and dependency
- Helps in code development and maintenance

In a To-Do List application, a class diagram includes classes such as Task, TaskManager, and StorageHandler, which define how tasks are created, managed, and stored.



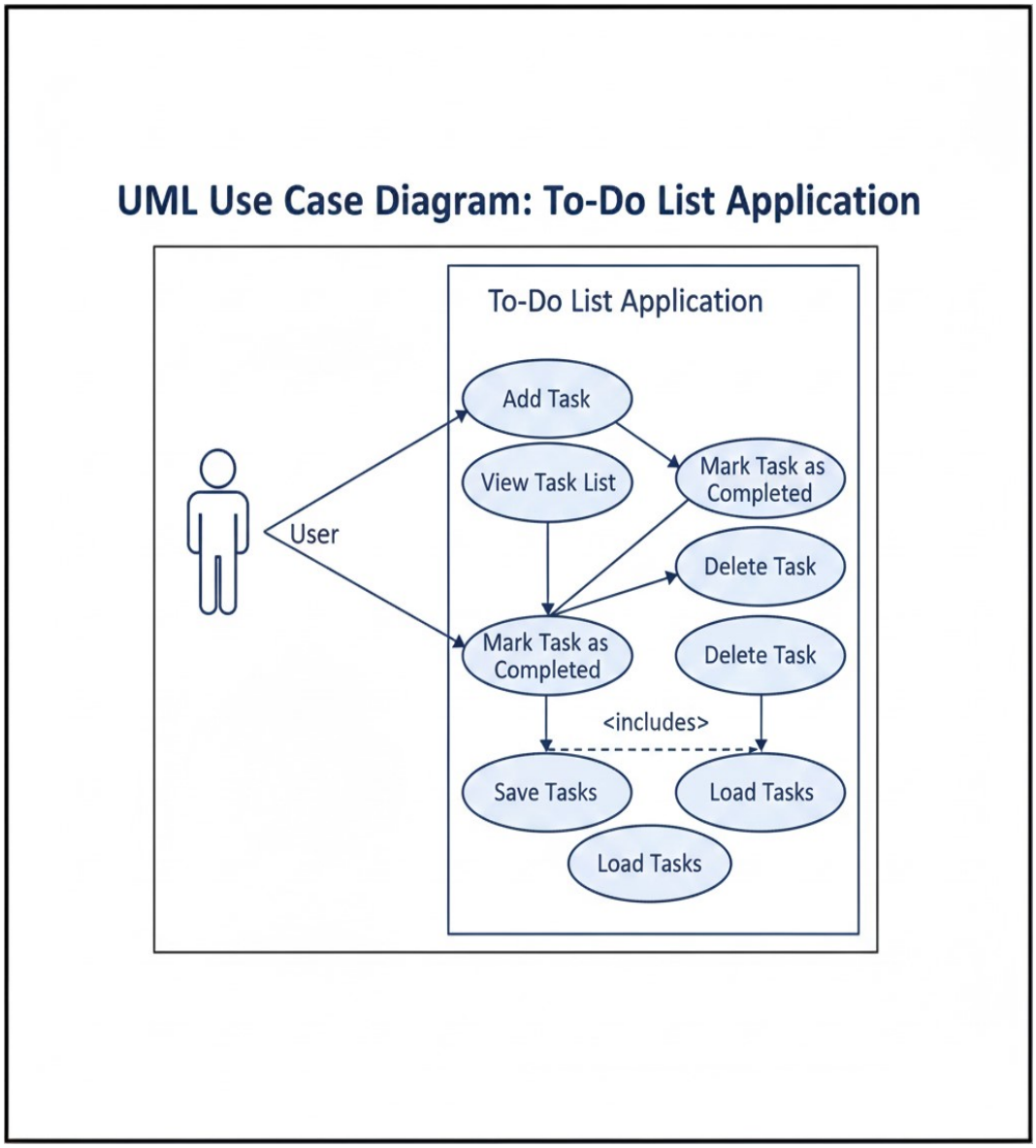
UML Use Case Diagram

A **Use Case Diagram** represents the **functional behavior** of a system from the user's perspective. It shows how users (actors) interact with the system to achieve specific goals.

Key features of a Use Case Diagram:

- Identifies actors and their interactions
- Represents system functionality using use cases
- Helps in requirement analysis and validation

In a To-Do List application, the user can perform actions such as adding tasks, viewing tasks, marking tasks as completed, and deleting tasks.



TECHNOLOGY DESCRIPTION

This section describes the technologies used to develop the application.

Front-End Technology

Java Swing

Java Swing is used to create the graphical user interface. It provides components such as buttons, text fields, and lists to build interactive applications.

Advantages

- Platform independent
- Rich UI components
- Easy integration with Java

Back-End Technology

Java Core

Core Java is used for implementing application logic, handling events, and managing data.

File Handling

Java file handling APIs are used to store tasks persistently.

DATABASE DESIGN

The application uses a **file-based storage system** instead of a traditional database.

Database Description

- File Name: `tasks.txt`
- Storage Type: Text file
- Data Format: One task per line

Advantages of File-Based Storage

- Simple to implement
- No additional setup required
- Suitable for small-scale applications

Limitations

- Not suitable for large-scale data
- No concurrent access support

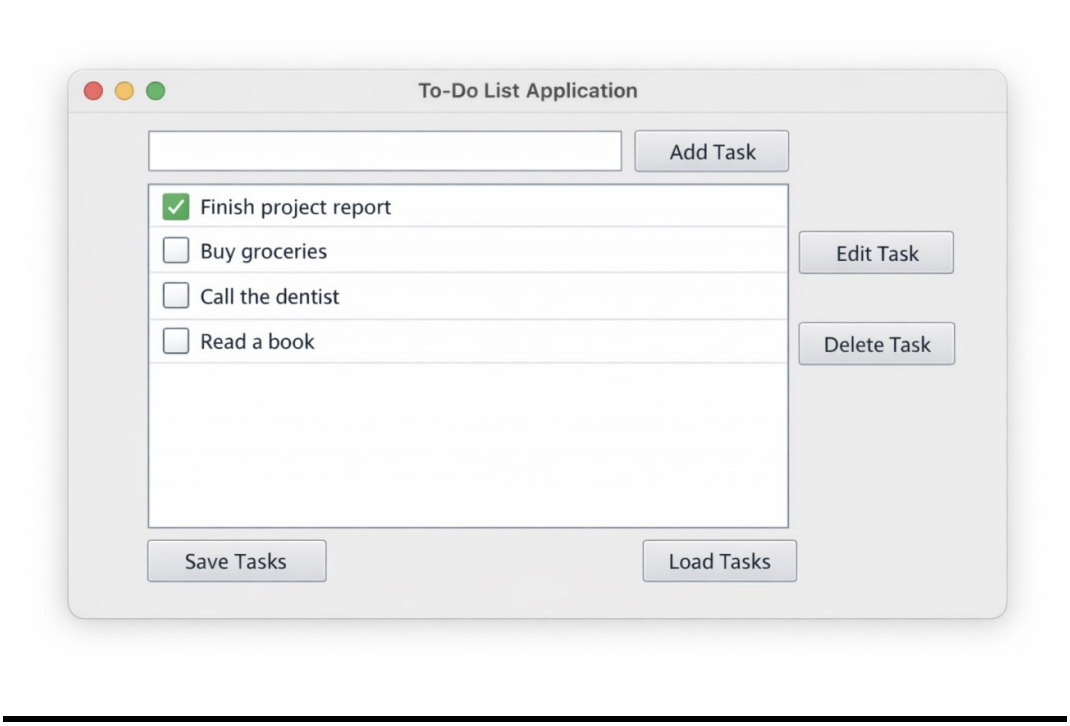
CODING

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.ArrayList;

public class TodoApp extends JFrame {

    private JTextField taskField;
    private JButton addButton, deleteButton;
    private DefaultListModel<String> listModel;
    private JList<String> taskList;

    private static final String FILE_NAME = "tasks.txt";

    public TodoApp() {
        setTitle("To Do List");
        setSize(450, 550);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Main container
        JPanel container = new JPanel();
        container.setBackground(new Color(21, 54, 119));
        container.setLayout(new BorderLayout(10, 10));
        container.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        // App panel (white card)
        JPanel appPanel = new JPanel();
        appPanel.setBackground(Color.WHITE);
        appPanel.setLayout(new BorderLayout(15, 15));
        appPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        // Title
        JLabel title = new JLabel("To Do List", SwingConstants.CENTER);
        title.setFont(new Font("Poppins", Font.BOLD, 24));
        title.setForeground(new Color(0, 39, 101));
        appPanel.add(title, BorderLayout.NORTH);

        // Input row
        JPanel inputPanel = new JPanel(new BorderLayout(10, 10));
        inputPanel.setBackground(new Color(200, 200, 200));
        inputPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        taskField = new JTextField();
        taskField.setFont(new Font("Arial", Font.PLAIN, 16));
        inputPanel.add(taskField, BorderLayout.CENTER);

        addButton = new JButton("ADD");
        addButton.setBackground(new Color(255, 89, 69));
```

```
        addButton.setForeground(Color.WHITE);
        addButton.setFocusPainted(false);
        inputPanel.add(addButton, BorderLayout.EAST);

        appPanel.add(inputPanel, BorderLayout.CENTER);

        // Task list
        listModel = new DefaultListModel<>();
        taskList = new JList<>(listModel);
        taskList.setFont(new Font("Arial", Font.PLAIN, 16));
        taskList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        JScrollPane scrollPane = new JScrollPane(taskList);
        appPanel.add(scrollPane, BorderLayout.SOUTH);

        // Delete button
        deleteButton = new JButton("DELETE SELECTED");
        deleteButton.setBackground(new Color(132, 150, 185));
        deleteButton.setForeground(Color.WHITE);
        deleteButton.setFocusPainted(false);

        container.add(appPanel, BorderLayout.CENTER);
        container.add(deleteButton, BorderLayout.SOUTH);

        add(container);

        // Load saved tasks
        loadTasks();

        // Add task
        addButton.addActionListener(e -> addTask());

        // Delete task
        deleteButton.addActionListener(e -> deleteTask());

        // Mark task completed on click
        taskList.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                int index = taskList.locationToIndex(e.getPoint());
                if (index >= 0) {
                    String task = listModel.get(index);
                    if (!task.startsWith("✔ ")) {
                        listModel.set(index, "✔ " + task);
                    }
                    saveTasks();
                }
            }
        });
    }

    private void addTask() {
        String task = taskField.getText().trim();
        if (task.isEmpty()) {
```

```
            JOptionPane.showMessageDialog(this, "You must write something");
            return;
        }
        listModel.addElement(task);
        taskField.setText("");
        saveTasks();
    }

    private void deleteTask() {
        int selectedIndex = taskList.getSelectedIndex();
        if (selectedIndex != -1) {
            listModel.remove(selectedIndex);
            saveTasks();
        }
    }

    private void saveTasks() {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(FILE_NAME))) {
            for (int i = 0; i < listModel.size(); i++) {
                writer.write(listModel.get(i));
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void loadTasks() {
        File file = new File(FILE_NAME);
        if (!file.exists()) return;

        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = reader.readLine()) != null) {
                listModel.addElement(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new TodoApp().setVisible(true);
        });
    }
}
```

OUTPUT


To-Do List Application
Add Task
Finish project report
Buy groceries
Call the dentist
Read a book
Edit Task
Delete Task
Save Tasks
Load Tasks

TESTING

Testing is an essential phase of the Software Development Life Cycle (SDLC) that ensures the Todo List application works correctly, efficiently, and reliably. The main objective of testing in this project is to verify that all features such as adding, updating, deleting, and managing tasks function as expected without errors.

In the Todo List mini project, different testing techniques were applied, including Unit Testing, Alpha Testing, Beta Testing, Integration Testing, and Object-Oriented Testing.

Unit Testing

Unit Testing involves testing individual components or functions of the Todo List application independently. Each module is tested in isolation to confirm that it performs its intended task correctly.

Objectives of Unit Testing:

- To verify the correctness of individual functions
- To detect errors at an early stage
- To ensure reliable task operations
- To simplify debugging and maintenance

Unit Testing in Todo List Project:

In this project, unit testing was performed on core functionalities such as:

- Adding a new task
- Editing an existing task
- Marking a task as completed
- Deleting a task
- Input validation (empty task, duplicate task)

Each function was tested with valid and invalid inputs to ensure correct behavior.

Alpha Testing

Alpha Testing is carried out by the developer in a controlled environment before releasing the application to users. This testing focuses on identifying major functional issues and user interface problems.

Objectives of Alpha Testing:

- To identify functional defects early
- To check overall workflow of the application
- To verify UI consistency and usability

Alpha Testing in Todo List Project:

During alpha testing, the Todo List application was tested internally by executing all features sequentially. Issues related to task input, button actions, and page responsiveness were identified and fixed. This ensured smooth task management before user-level testing.

Beta Testing

Beta Testing is performed by real users in a real-world environment. It helps identify usability issues and collect feedback for improvement.

Objectives of Beta Testing:

- To evaluate application behavior with real users
- To identify usability and performance issues
- To collect user feedback

Beta Testing in Todo List Project:

In this project, beta testing was conducted by a small group of users. Users interacted with the application to manage their daily tasks and provided feedback. Based on the feedback, minor improvements were made, such as enhancing UI clarity and improving task completion indicators.

Integration Testing

Integration Testing verifies the interaction between different modules of the Todo List application to ensure smooth data flow.

Objectives of Integration Testing:

- To ensure proper communication between frontend and backend
- To validate data storage and retrieval
- To detect interface-level defects

Integration Testing in Todo List Project:

Integration testing was performed to verify interactions between:

- User interface and task logic
- Task operations and database/local storage

Scenarios such as adding a task and immediately viewing it, updating task status, and deleting tasks were tested to ensure seamless integration.

Object-Oriented Testing

Object-Oriented Testing focuses on testing classes, objects, and their interactions in an object-oriented system.

Objectives of Object-Oriented Testing:

- To verify class behavior and method correctness
- To test object interactions
- To ensure data encapsulation

Object-Oriented Testing in Todo List Project:

In the Todo List application, object-oriented testing was applied by testing classes such as Task and TaskManager. Methods related to task creation, update, deletion, and status management were tested to ensure correct object behavior. This improved code reliability and maintainability.

GANTT CHART

A Gantt Chart is a project management tool used to represent the project schedule visually. It shows the list of project activities along with their start time, end time, and duration. The Gantt Chart helps in planning, scheduling, tracking progress, and ensuring timely completion of the project.

In this Todo List mini project, the Gantt Chart was used to plan different phases such as requirement analysis, design, development, testing, and documentation.

Objectives of Gantt Chart:

- To plan project activities systematically
- To allocate time for each phase of the project
- To track progress and deadlines
- To ensure timely completion of the project
-

Gantt Chart for Todo List Mini Project:

Project Activity	Duration	Time Period
Requirement Analysis	2 Days	Week 1
System Design	2 Days	Week 1
Frontend Development	4 Days	Week 2
Backend / Logic Development	4 Days	Week 2
Integration	2 Days	Week 3
Testing	3 Days	Week 3
Debugging & Optimization	2 Days	Week 4
Documentation	2 Days	Week 4

Description:

The Gantt Chart illustrates that the Todo List project was completed in a structured manner over a period of four weeks. Each phase was carefully planned and executed sequentially. Development and testing phases were given more time to ensure correct functionality and application stability.

CONCLUSION

The To-Do List Application successfully fulfills its objective of providing a simple and effective task management solution. The project demonstrates the application of software engineering principles, system analysis, design, and implementation. It improves productivity and task organization while serving as a strong academic mini-project.

BIBLIOGRAPHY

1. Pressman, R. S. *Software Engineering: A Practitioner's Approach*
2. Java Documentation – Oracle
3. Software Engineering Notes
4. Online Java Tutorials