Programming 3
BMEVIIIAB00

To-Do List Application

a minimalistic task manager with a robust gui

MUHAMMAD IBRAHIM SHOEB

BSc in Computer Science Engineering, BME
OZLVV3

github.com/ibrahimify/TodoListApp

TABLE OF CONTENT

INTRODUCTION

The **To-Do List Application** is a Java desktop application that allows users to manage their daily tasks efficiently. The application provides a graphical user interface (GUI) built with **Java Swing**, featuring functionality for creating, editing, deleting, categorizing, and sorting tasks. Tasks can be filtered by their status or category, and the user interface supports both **light and dark modes**. Data is persistently stored using **Java serialization**, ensuring that tasks and categories are saved between sessions.

The objective of this homework is to practice key concepts such as:

- **Swing-based GUI** for task management.
- **Collections framework** for managing tasks and categories.
- **Java Serialization** for saving and loading tasks.
- **JUnit Unit Testing** to validate the functionality of the system.

Technology Used

- **Java 21**
- **Java Swing** for the graphical interface (GUI)
- **Java Serialization** for data persistence
- **JUnit** for unit testing
- **Maven** for project management

PROGRAM INTERFACE

User Interface (UI) Overview

The application features a clean and simple GUI built using **Java Swing**. The main components are:

- **Task List**: A list displaying tasks, including options to add, edit, or delete tasks.
- **Category Management**: A panel to view and manage categories.
- **Menu Bar**: For accessing settings like switching between light/dark modes.

How to Start the Program

To start the program:

1. Ensure you have **Java 17** or higher installed.
2. Build and run the application using Maven or from the IDE.

How to Terminate the Program:

1. Clicking the close button on the window.

PROGRAM EXECUTION

User Interaction:

The program is driven by user interactions with the GUI. Users can:

- Add new tasks.
- Edit and delete existing tasks.
- Organize tasks into categories.
- Filter tasks based on completion status or category.

Task Management

Tasks are represented by the **Task** class. Each task has a title, description, due date, and status. Tasks can be marked as completed, edited, or deleted.

Task Filtering and Sorting

- **Completion Status**: Show completed or pending tasks.
- **Category**: Show tasks belonging to a specific category.

Data Persistence

Tasks and categories are saved and loaded from a file using **Java serialization**. This ensures the data persists between application sessions.

INPUT AND OUTPUT

Input

- User input is collected via **text fields** (for titles, descriptions, etc.) and **date pickers** (for due dates).
- Categories are chosen from a list of predefined categories or can be created dynamically.

Output

- The application outputs task details in a list format, with sorting and filtering options.

- The application also outputs success/error messages in case of invalid input or data issues.

Data Persistence

Tasks and categories are saved in a **serialized file** (**categories.ser**). When the program starts, it attempts to load this data, allowing users to resume their work.

PROGRAM STRUCTURES

Overview:

The program is structured into several key classes:

- **Task**: Represents a single task with properties such as title, description, and due date.
- **Category**: Represents a category to which tasks can belong.
- **TaskManager**: Manages task data persistence and loading/saving tasks.
- **AddTaskDialog**: Manages task creation.
- **EditTaskDialog**: Manages task editing.
- **TodoAppGUI**: The graphical interface that integrates all components.
- **Main**: The entry point of the program.

Task Class

Manages the details of a task, including title, description, due date, completion status, and category.

Category Class

Manages task categories, allowing tasks to be grouped.

TaskManager Class

Handles saving and loading tasks and categories using **Java serialization**.

AddTaskDialog and EditTaskDialog Classes

These classes handle the creation and modification of tasks, respectively.

TodoAppGUI Class

This class provides the **GUI**, enabling users to interact with the application.

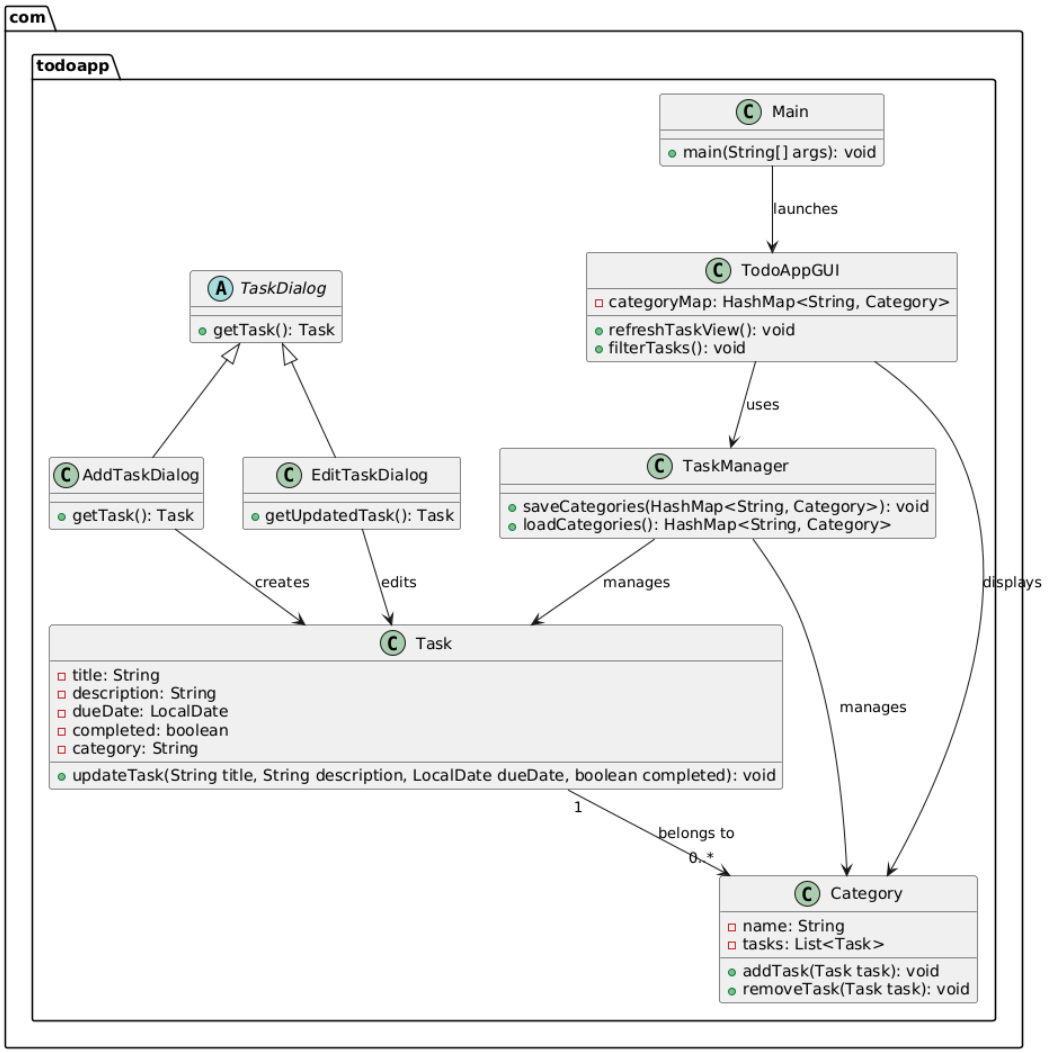
Main Class

The entry point of the program, which initializes and runs the application.

CLASS DIAGRAMS AND SEQUENCE DIAGRAMS

Class Diagram

The class diagram represents the Todo List Application's architecture, showcasing key components and their interactions:

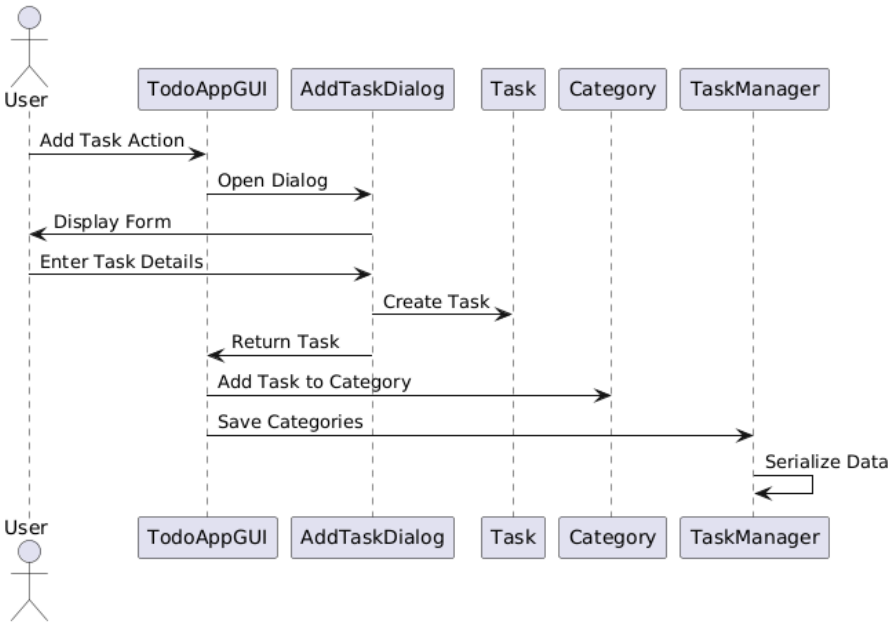


Sequence Diagrams for Key Use-Cases

These sequence diagrams represent the interactions for core tasks in the application, such as adding, editing, and deleting tasks.

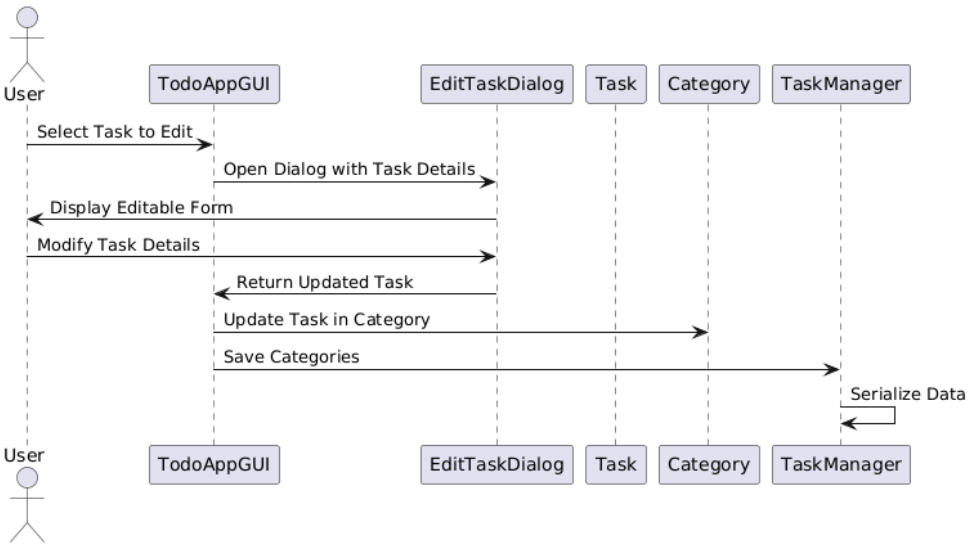
1. **Add Task**

   The flow for adding a new task.



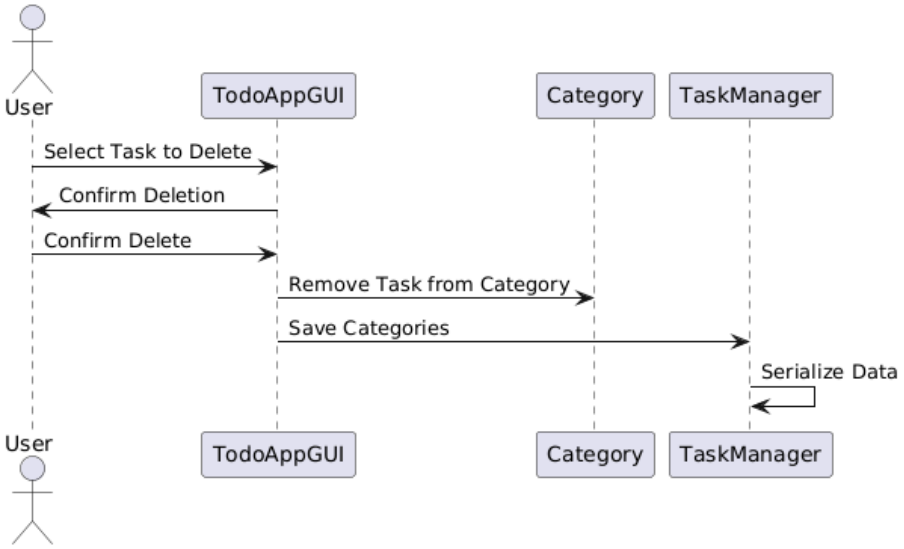
2. **Edit Task**

   The flow for editing an existing task.



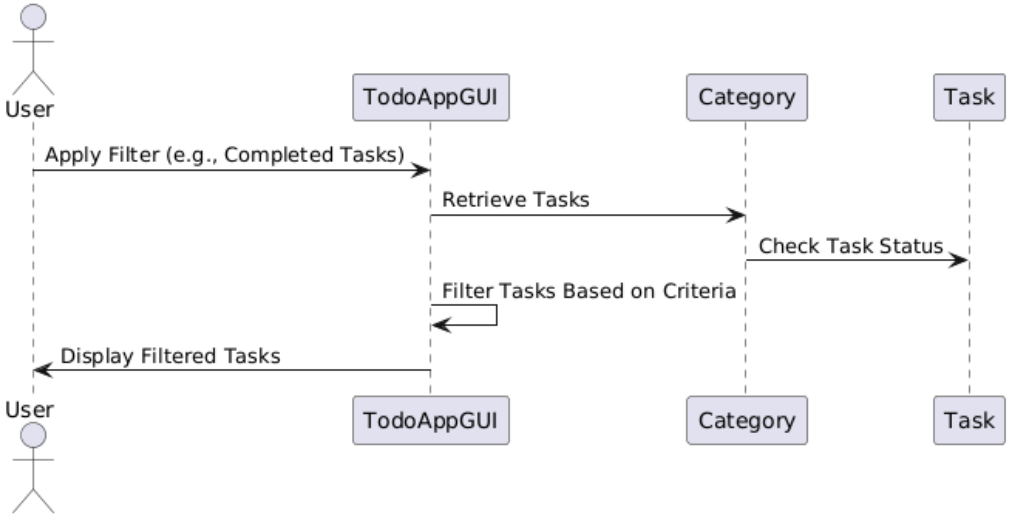
3. **Delete Task**

   The flow for deleting a task.



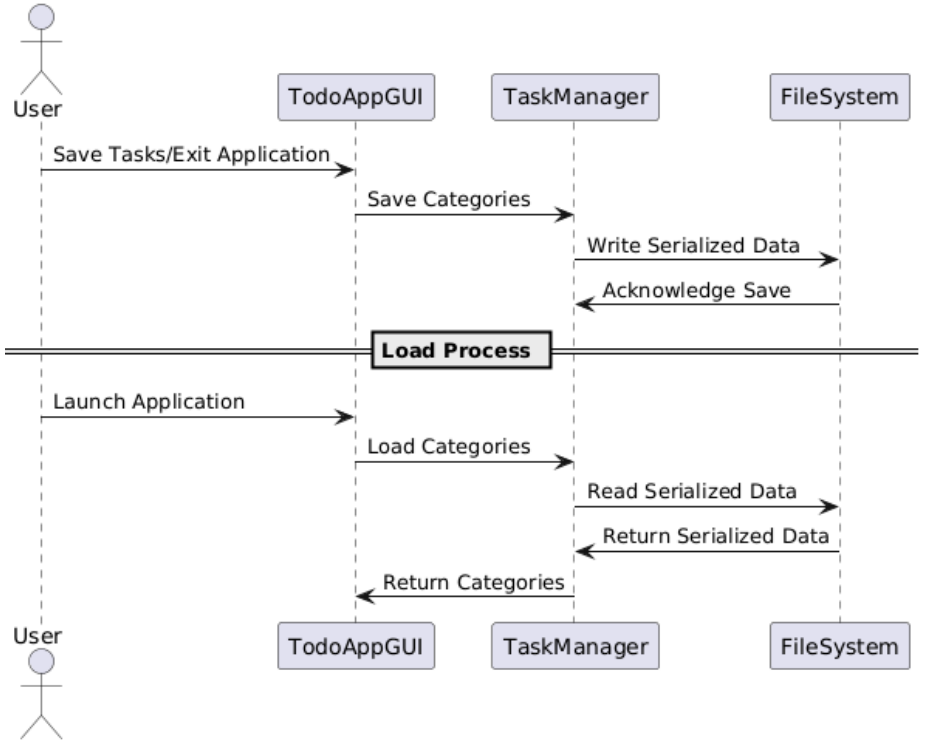
4. **Filter Tasks**

   Filtering tasks based on criteria.



5. **Save and Load Tasks**

   Flow for saving and loading tasks via serialization.



TESTING AND VERIFICATION

Unit Testing Overview

Unit tests are written using **JUnit** and ensure the functionality of key components such as task creation, editing, and data persistence.

Test Cases Implemented

1. TaskTest.java

testTaskCreation()

- **Purpose**: Verifies that a task is correctly created with the specified attributes.
- **What it Tests**: Ensures the task's title, description, due date, completion status, and category are set properly during creation.
- **Key Assertion**: Checks that the task's properties match the input values.

testTaskEditing()

- **Purpose**: Verifies that a task can be edited after creation.
- **What it Tests**: Validates that task properties like title, description, due date, completion status, and category can be updated correctly.

- **Key Assertion**: Ensures that the task's edited properties reflect the new values.

---

2. SortingAndFilteringTest.java

testSortingByTitle()

- **Purpose**: Verifies that tasks can be sorted by title.
- **What it Tests**: Checks the sorting functionality based on the task's title in alphabetical order.
- **Key Assertion**: Confirms that the tasks are sorted correctly.

testFilteringByCompletionStatus()

- **Purpose**: Verifies that tasks can be filtered by completion status.
- **What it Tests**: Ensures that only completed tasks are included when filtering by completion status.
- **Key Assertion**: Confirms that the list contains only completed tasks.

---

3. TaskManagerTest.java

testSaveAndLoadCategories()

- **Purpose**: Verifies that categories and tasks are saved and loaded correctly using Java serialization.
- **What it Tests**: Ensures that categories and tasks are correctly serialized to a file and deserialized on load.
- **Key Assertion**: Validates that the loaded categories and tasks match the saved data.

Test Results

All tests pass successfully, ensuring that the application works as expected.

IMPROVEMENTS AND EXTENSIONS

Planned Features

Future enhancements could include:

- Adding **task priority** functionality.

- **Notifications** for task due dates.
- Adding a **search bar** for quick task lookups.

Potential Enhancements

- **UI Enhancements**: Expanding the UI with more customization options for task sorting or additional features like task priorities.
- **Mobile Support**: Developing a mobile version of the app for cross-platform compatibility.
- Implement **cloud synchronization** for task data.

DIFFICULTIES ENCOUNTERED

During development, challenges included:

- Implementing Java Serialization effectively for data persistence.
- Designing a clear and intuitive user interface using Java Swing.
- Ensuring smooth interaction between the various components of the application (task management, GUI, and persistence).

CONCLUSION

In conclusion, the **To-Do List Application** serves as a practical example of Java Swing and serialization. It showcases the application of essential Java concepts such as GUI development, data handling, and file persistence. The project meets the objectives outlined in the homework, fulfilling the requirements for task management, GUI functionality, and testing.

REFERENCES

[1] Balaguruswamy, E. 2007. *Programming with JAVA - A Primer: Third Edition*. Third Edition ed. New Delhi, India: Tata McGraw-Hill Publishing Company Limited.

[2] “Java Documentation - Get Started.” n.d. Oracle Help Center. Accessed December 4, 2024. https://docs.oracle.com/en/java.

[3] “JUnit 5 User Guide.” n.d. JUnit 5. Accessed December 4, 2024. https://junit.org/junit5/docs/current/user-guide/.