NATION UNIVERSITY OF MODERN LANGUAGES

DEPARTMENT OF SOFTWARE ENGINEERING

IMPLEMENTATION & TESTING

GITHUB REPOSITORY

SUBMITTED BY	**M. MUZAMMIL KHAN (SP21401)**
	ABDUL GHAFOOR (SP21375)
	M. BURHAN (SP21378)
GROUP NAME	**DIGITAL EMPIRE**
SUBMITTED TO	**MS. FATIMA GILLANI**
SUBJECT	**SCD**
SECTION	**BSSE-A-AFTERNOON**
SEMESTER	**5th**

Task Management Application

CODE

Class 1:

```
class Task {
    private String description;
    private boolean completed;
    private TimeTracker time_tracker;

    public Task(String description) {
        this.description = description;
        this.completed = false;
        this.time_tracker = new TimeTracker();
        this.time_tracker.record_creation_time();
    }
//returns description of new task
    public String get_description() {
        return description;
    }
 // Returns true if the task is completed, otherwise returns false.
    public boolean is_completed() {
        return completed;
    }

    public void mark_as_completed() {
        this.completed = true;
        this.time_tracker.record_completion_time();
    }

    public String get_completion_time() {
        return time_tracker.get_completion_time();
    }

    public String get_duration() {
        return time_tracker.get_duration();
    }

}
```

Class 2:

```
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class TimeTracker {
    private LocalDateTime creation_time;
    private LocalDateTime completion_time;
```

```
    public void record_creation_time() {
        this.creation_time = LocalDateTime.now();
    }

    public void record_completion_time() {
        this.completion_time = LocalDateTime.now();
    }

    public String get_completion_time() {
        try {
            if (completion_time != null) {
                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                return completion_time.format(formatter);
            }
            return "Not completed";
        } catch (Exception e) {
            return "Error retrieving completion time";
        }
    }

    public String get_duration() {
        try {
            if (completion_time != null) {
                Duration duration = Duration.between(creation_time,
completion_time);
                long seconds = duration.getSeconds();
                long hours = seconds / 3600;
                seconds %= 3600;
                long minutes = seconds / 60;
                seconds %= 60;
                return String.valueOf(hours) + ":" +
String.valueOf(minutes) + ":" + String.valueOf(seconds);
            }
            return "Task not completed";
        } catch (Exception e) {
            return "Error calculating duration";
        }
    }

}
```

Class 3:

```
import java.util.ArrayList;

class TaskManager {
    private TaskListManager to_do_list_manager;
    private TaskListManager in_progress_list_manager;
    private TaskListManager completed_list_manager;

    public TaskManager() {
```

```
        to_do_list_manager = new TaskListManager();
        in_progress_list_manager = new TaskListManager();
        completed_list_manager = new TaskListManager();
    }

    public void add_task(Task task) {
        try {
            if (task != null) {
                to_do_list_manager.add_task(task);
            } else {
                throw new IllegalArgumentException("Task cannot be null");
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public void move_task_to_in_progress(int index) {
        Task task = to_do_list_manager.remove_task(index);
        if (task != null) {
            in_progress_list_manager.add_task(task);
        }
    }

    public void move_task_to_completed(int index) {
        Task task = in_progress_list_manager.remove_task(index);
        if (task != null) {
            task.mark_as_completed();
            completed_list_manager.add_task(task);
        }
    }
    // Returns the to-do list of tasks
    public ArrayList<Task> get_to_do_list() {
        return to_do_list_manager.get_tasks();
    }
    // Returns the in-progress list of tasks
    public ArrayList<Task> get_in_progress_list() {
        return in_progress_list_manager.get_tasks();
    }
 // Returns the completed list of tasks
    public ArrayList<Task> get_completed_list() {
        return completed_list_manager.get_tasks();
    }
}
```

Class 4:

```
import java.util.ArrayList;

public class TaskListManager {

    public ArrayList<Task> tasks;

    public TaskListManager() {
```

```
        tasks = new ArrayList<>();
    }

    public void add_task(Task task) {
        tasks.add(task);
    }
 // Method to remove a task from the task list based on its index
    public Task remove_task(int index) {
        if (index >= 0 && index < tasks.size()) {
            return tasks.remove(index);
        }
        return null;
    }
    // Method to retrieve all tasks in the task list
    public ArrayList<Task> get_tasks() {
        return tasks;
    }
}
```

Class 5:

```
import javax.swing.*;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TaskManagementGUI extends JFrame implements ActionListener {
    private TaskManager task_manager;
    private JTextPane to_do_area;
    private JTextPane in_progress_area;
    private JTextPane completed_area;
    private JButton add_task_button;
    private JButton move_in_progress_button;
    private JButton move_completed_button;
    private JButton exit_button;

    public TaskManagementGUI() {
        task_manager = new TaskManager();
        setTitle("Task Management Application");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel main_panel = new JPanel(new GridLayout(1, 3));
```

```
     // Text panes for displaying tasks
        to_do_area = new JTextPane();
        in_progress_area = new JTextPane();
        completed_area = new JTextPane();

     // Scroll panes for task list text panes
        JScrollPane to_do_scroll_pane = new JScrollPane(to_do_area);
        JScrollPane in_progress_scroll_pane = new
JScrollPane(in_progress_area);
        JScrollPane completed_scroll_pane = new
JScrollPane(completed_area);

        add_task_button = new JButton("Add Task");
        move_in_progress_button = new JButton("Move to In Progress");
        move_completed_button = new JButton("Move to Completed");
        exit_button = new JButton("Exit"); // Instantiate the exit button

        add_task_button.addActionListener(this);
        move_in_progress_button.addActionListener(this);
        move_completed_button.addActionListener(this);
        exit_button.addActionListener(this);

        JPanel button_panel = new JPanel(new GridLayout(4, 1));
        button_panel.add(add_task_button);
        button_panel.add(move_in_progress_button);
        button_panel.add(move_completed_button);
        button_panel.add(exit_button);

        main_panel.add(create_panel_with_label(to_do_scroll_pane, "To Do",
new Font("Arial", Font.BOLD, 30), Color.BLUE));
        main_panel.add(create_panel_with_label(in_progress_scroll_pane,
"In Progress", new Font("Arial", Font.BOLD, 30), Color.RED));
        main_panel.add(create_panel_with_label(completed_scroll_pane,
"Completed", new Font("Arial", Font.BOLD, 30), Color.GREEN));

        add(main_panel, BorderLayout.CENTER);
        add(button_panel, BorderLayout.EAST);
    }
 // Method to create a panel with a label of task list name
    private JPanel create_panel_with_label(Component component, String
label, Font font, Color color) {
        JPanel panel = new JPanel(new BorderLayout());
        JLabel title_label = new JLabel(label, SwingConstants.CENTER);
        title_label.setFont(font);
        title_label.setForeground(color);
        panel.add(title_label, BorderLayout.NORTH);
        panel.add(component, BorderLayout.CENTER);
        return panel;
    }
 // Method to update the task lists displayed on the GUI
    public void update_lists() {
        to_do_area.setText("");
        in_progress_area.setText("");
        completed_area.setText("");
```

```
        Font list_font = new Font("Arial", Font.PLAIN, 24);
        Color list_color = Color.BLACK;

        // Displaying tasks in each list
        int todo_count = 1;
        for (Task task : task_manager.get_to_do_list()) {
            append_text_with_font_and_color(to_do_area, todo_count++ + ".
" + task.get_description() + " ", list_font, list_color);
            add_button(to_do_area, task);
            append_new_line(to_do_area);
        }

        int in_progress_count = 1;
        for (Task task : task_manager.get_in_progress_list()) {
            append_text_with_font_and_color(in_progress_area,
in_progress_count++ + ". " + task.get_description() + " ", list_font,
list_color);
            add_button(in_progress_area, task);
            append_new_line(in_progress_area);
        }

        int completed_count = 1;
        for (Task task : task_manager.get_completed_list()) {
            append_text_with_font_and_color(completed_area,
completed_count++ + ". " + task.get_description() + " ", list_font,
list_color);
            add_button(completed_area, task);
            append_new_line(completed_area);
        }
    }

    private void append_new_line(JTextPane text_pane) {
        text_pane.replaceSelection("\n");
    }

    private void append_text_with_font_and_color(JTextPane text_pane,
String text, Font font, Color color) {
        SimpleAttributeSet attribute_set = new SimpleAttributeSet();
        StyleConstants.setFontFamily(attribute_set, font.getFamily());
        StyleConstants.setFontSize(attribute_set, font.getSize());
        StyleConstants.setForeground(attribute_set, color);
        text_pane.setCharacterAttributes(attribute_set, false);
        text_pane.replaceSelection(text);
    }
 // Method to add a button to a text pane for a task
    private void add_button(JTextPane text_pane, Task task) {
        TaskButtonHandler button_handler = new TaskButtonHandler(this,
task_manager, task);
        button_handler.putClientProperty("task", task);
        text_pane.insertComponent(button_handler);
    }

    @Override
```

```
    public void actionPerformed(ActionEvent e) {
        try {
            if (e.getSource() == add_task_button) {
                String description = JOptionPane.showInputDialog("Enter
task description:");
                if (description != null && !description.isEmpty()) {
                    Task new_task = new Task(description);
                    task_manager.add_task(new_task);
                    update_lists();
                }
            } else if (e.getSource() == move_in_progress_button) {
                String selected_text = to_do_area.getSelectedText();
                if (selected_text != null && !selected_text.isEmpty()) {
                    int index =
to_do_area.getText().indexOf(selected_text);
                    task_manager.move_task_to_in_progress(index);
                    update_lists();
                } else {
                  JOptionPane.showMessageDialog(this, "Please select a
task from the To Do list.", "Error", JOptionPane.ERROR_MESSAGE);
                }
            } else if (e.getSource() == move_completed_button) {
                String selected_text = in_progress_area.getSelectedText();
                if (selected_text != null && !selected_text.isEmpty()) {
                    int index =
in_progress_area.getText().indexOf(selected_text);
                    task_manager.move_task_to_completed(index);
                    update_lists();
                } else {
                  JOptionPane.showMessageDialog(this, "Please select a
task from the In Progress list.", "Error", JOptionPane.ERROR_MESSAGE);
                }
            } else if (e.getSource() == exit_button) {
                int confirm = JOptionPane.showConfirmDialog(this, "Are you
sure you want to exit?", "Exit", JOptionPane.YES_NO_OPTION);
                if (confirm == JOptionPane.YES_OPTION) {
                    System.exit(0);
                }
            }
        } catch (Exception ex) {
            System.out.println("Error: " + ex.getMessage());
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                TaskManagementGUI task_management_GUI = new
TaskManagementGUI();
                task_management_GUI.setVisible(true);
            }
        });
    }}
```

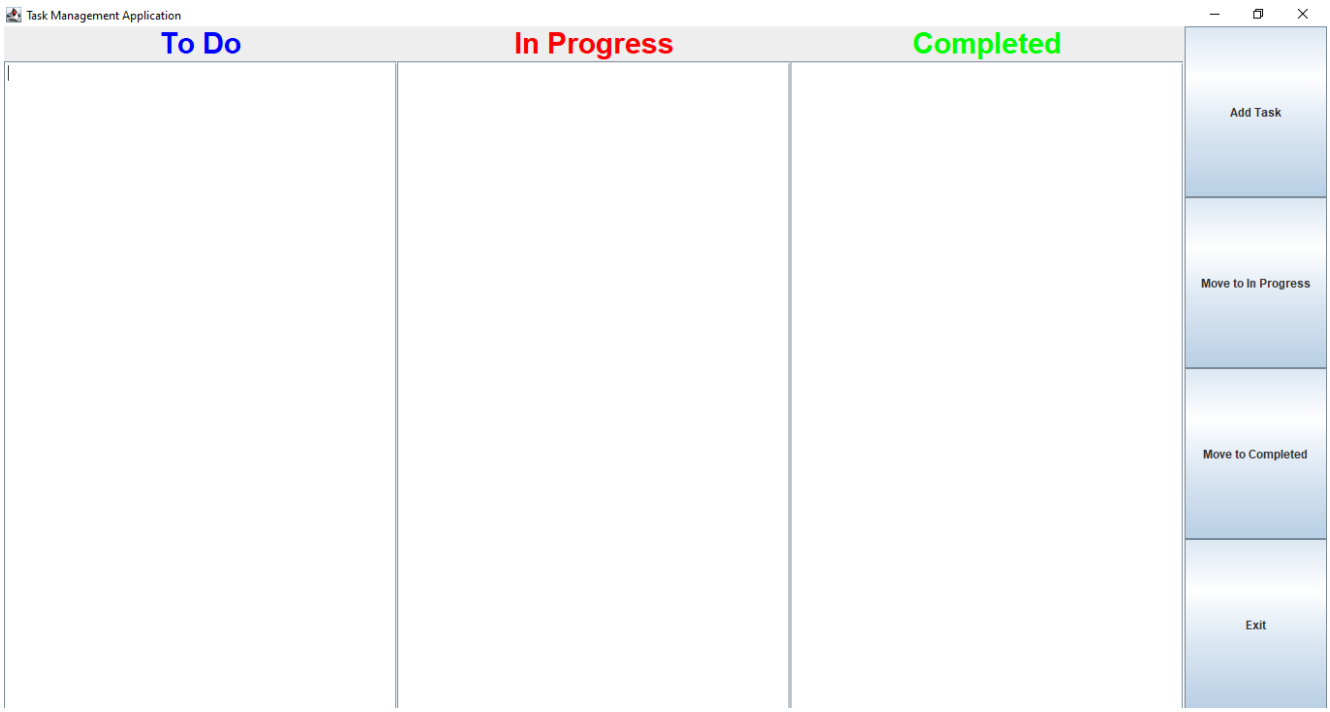
Class 6:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TaskButtonHandler extends JPanel implements ActionListener {
    private TaskManager task_manager;
    private Task task;
    private TaskManagementGUI gui;
    public TaskButtonHandler(TaskManagementGUI gui, TaskManager
task_manager, Task task) {
        this.gui = gui;
        this.task_manager = task_manager;
        this.task = task;

        setLayout(new FlowLayout(FlowLayout.RIGHT));

        JButton delete_button = new JButton("Delete");
        delete_button.addActionListener(this);
        add(delete_button);

        JButton details_button = new JButton("Details");
        details_button.addActionListener(this);
        add(details_button);
    }

@Override
    public void actionPerformed(ActionEvent e) {
        try {
            if (e.getActionCommand().equals("Delete")) {
                handle_delete();
            } else if (e.getActionCommand().equals("Details")) {
                handle_details();
            }
        } catch (Exception ex) {
            System.out.println("Error: " + ex.getMessage());
        }
    }
    private void handle_delete() {
        if (task.is_completed()) {
            task_manager.get_completed_list().remove(task);
        } else {
            if (task_manager.get_to_do_list().contains(task)) {
                task_manager.get_to_do_list().remove(task);
            } else if (task_manager.get_in_progress_list().contains(task))
{
                task_manager.get_in_progress_list().remove(task);
            }
        }
     // Updating the GUI after deletion
        gui.update_lists();
    }
```

```
    private void handle_details() {
        JOptionPane.showMessageDialog(this, "Task Completed at: " +
task.get_completion_time() + "\nTask Duration: " + task.get_duration());
    }
}
```

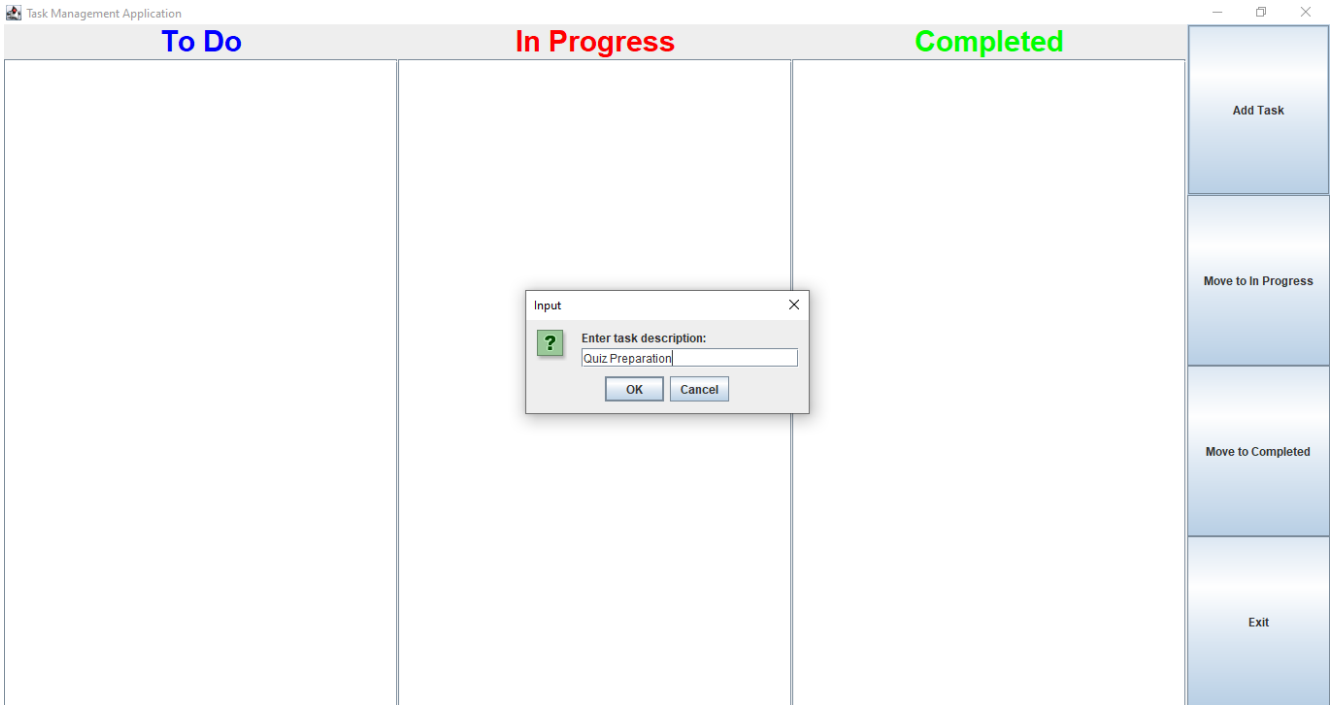
OUTPUT:

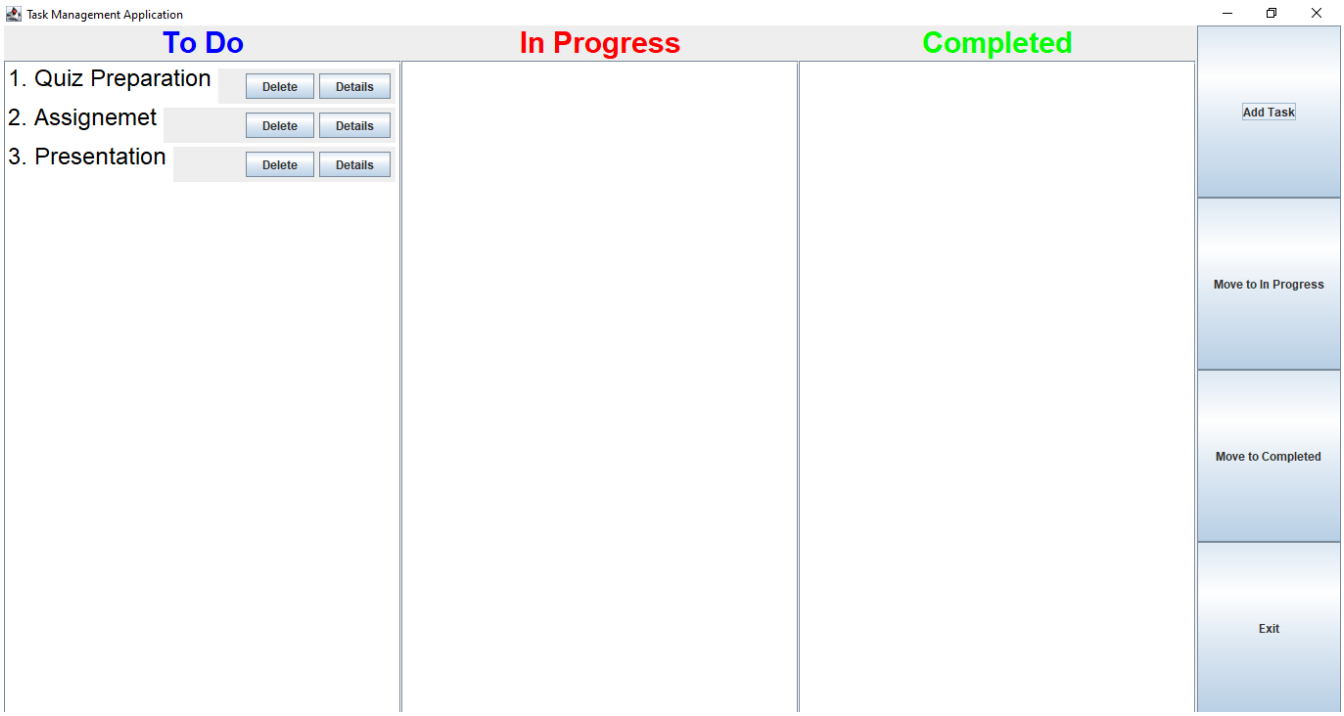
Task Management Application Interface



Step 1: Add Task

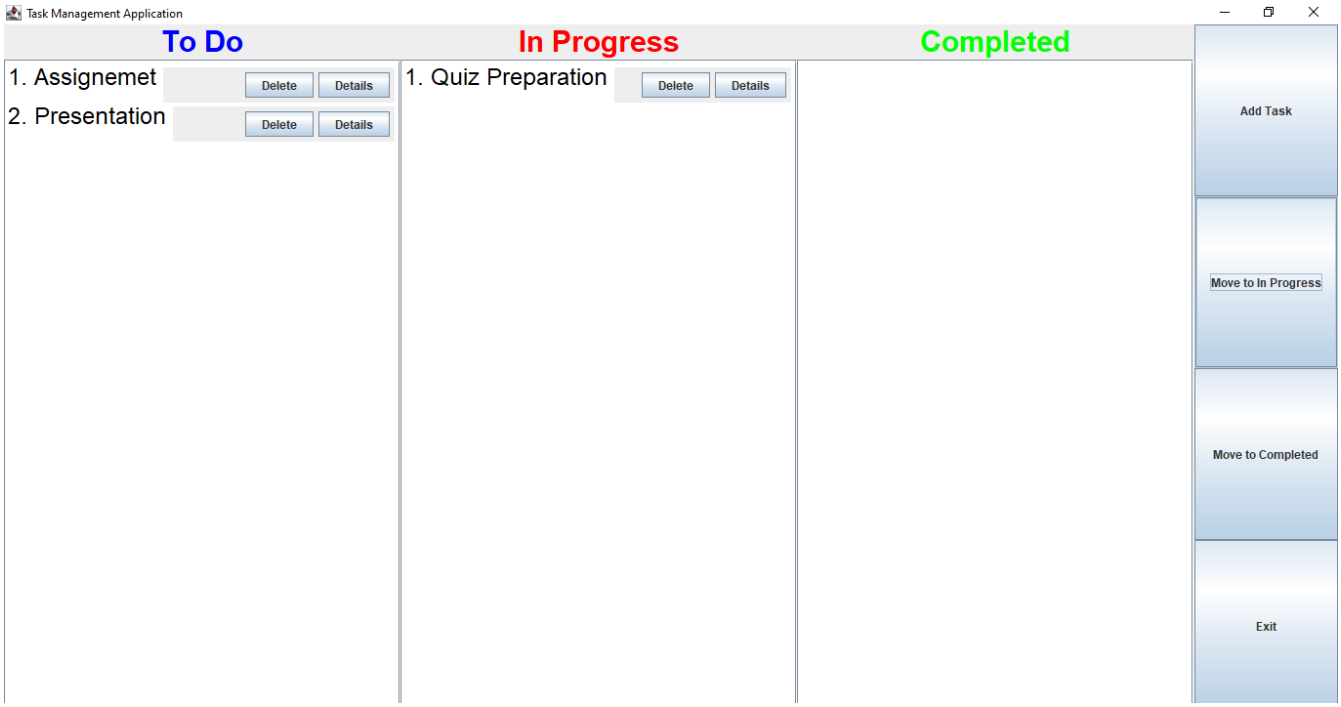
Click on add task button to add new task in Do-To List.





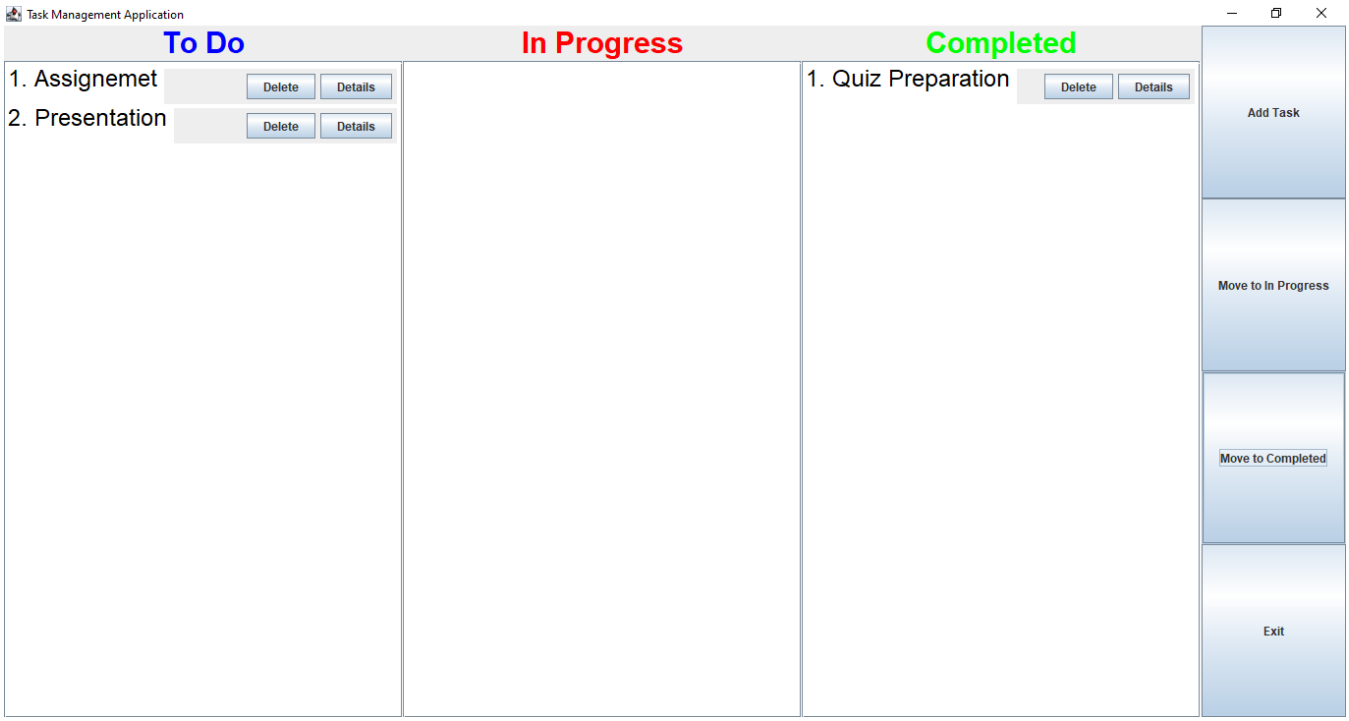
Step 2: Move Task to In Progress

When you start working on the task then select task from To-Do List and move it to In Progress by click the Move to In Progress button.



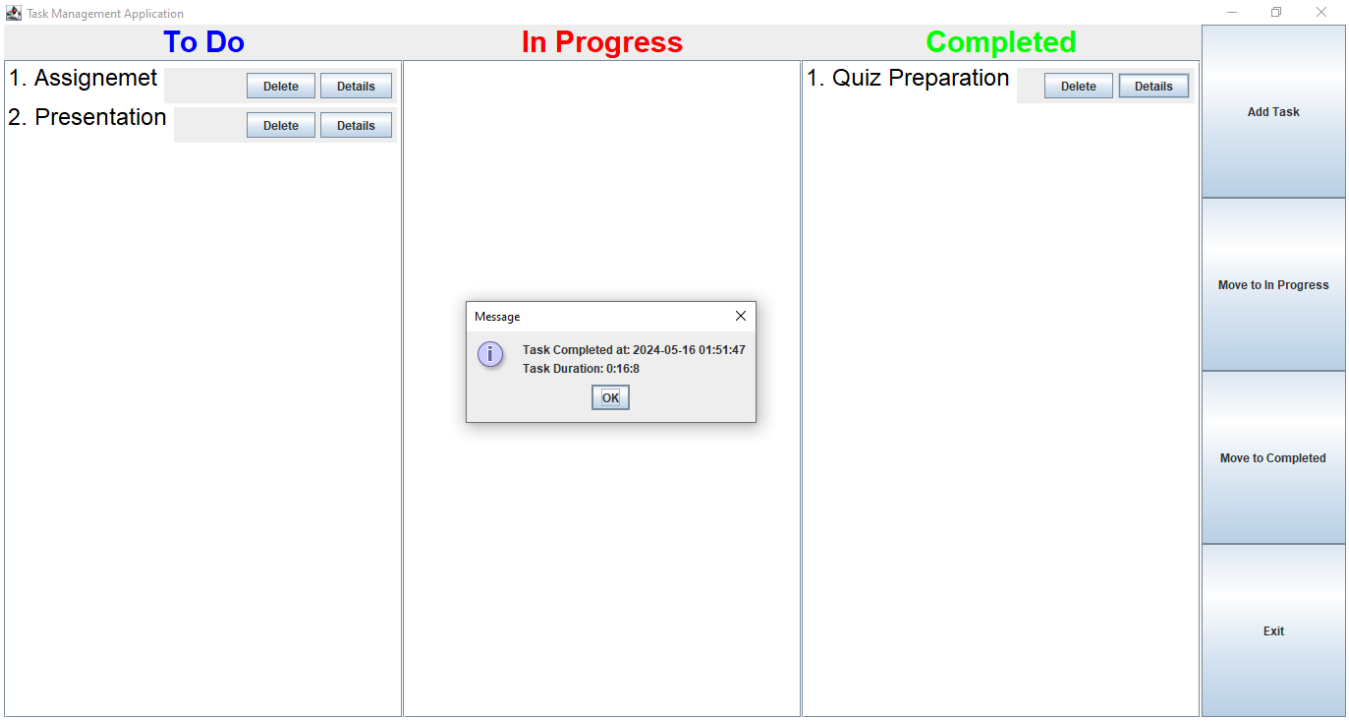
Step 3: Move to Completed

When you complete the task then select the task from In Progress and click the Move to Completed button to mark the task as completed.



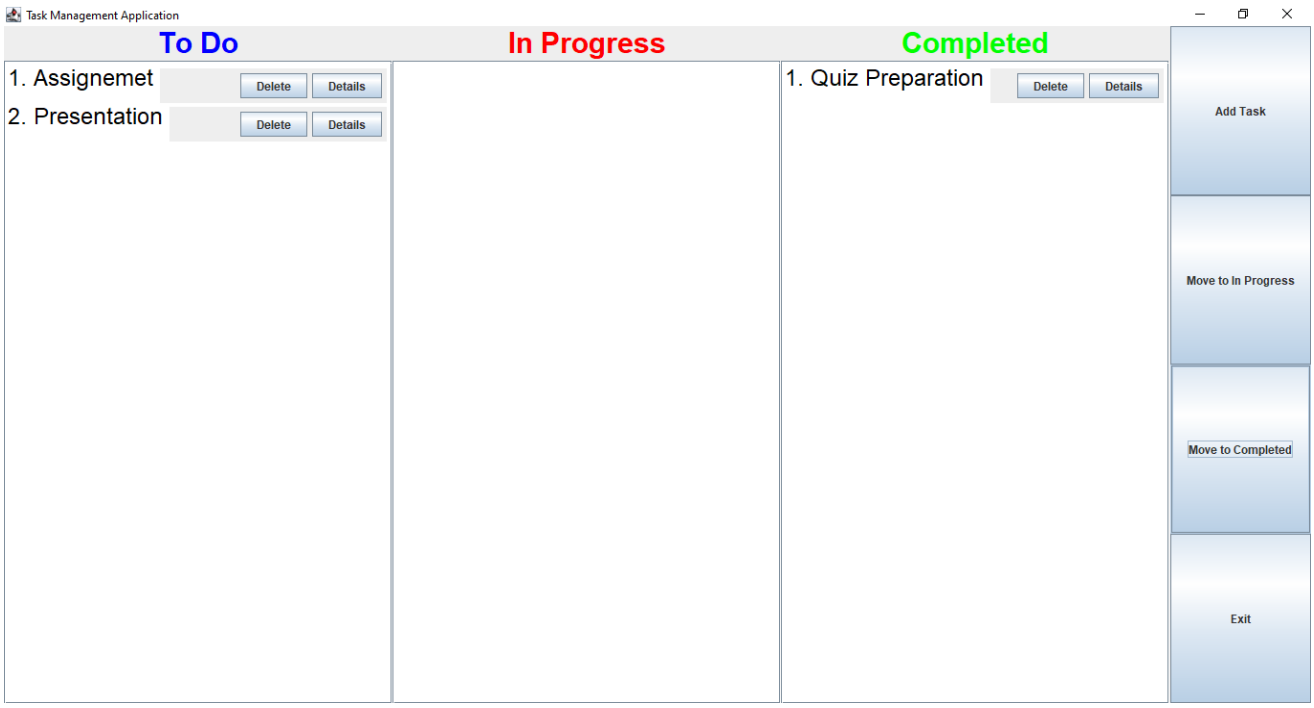
Step 4: Task Details

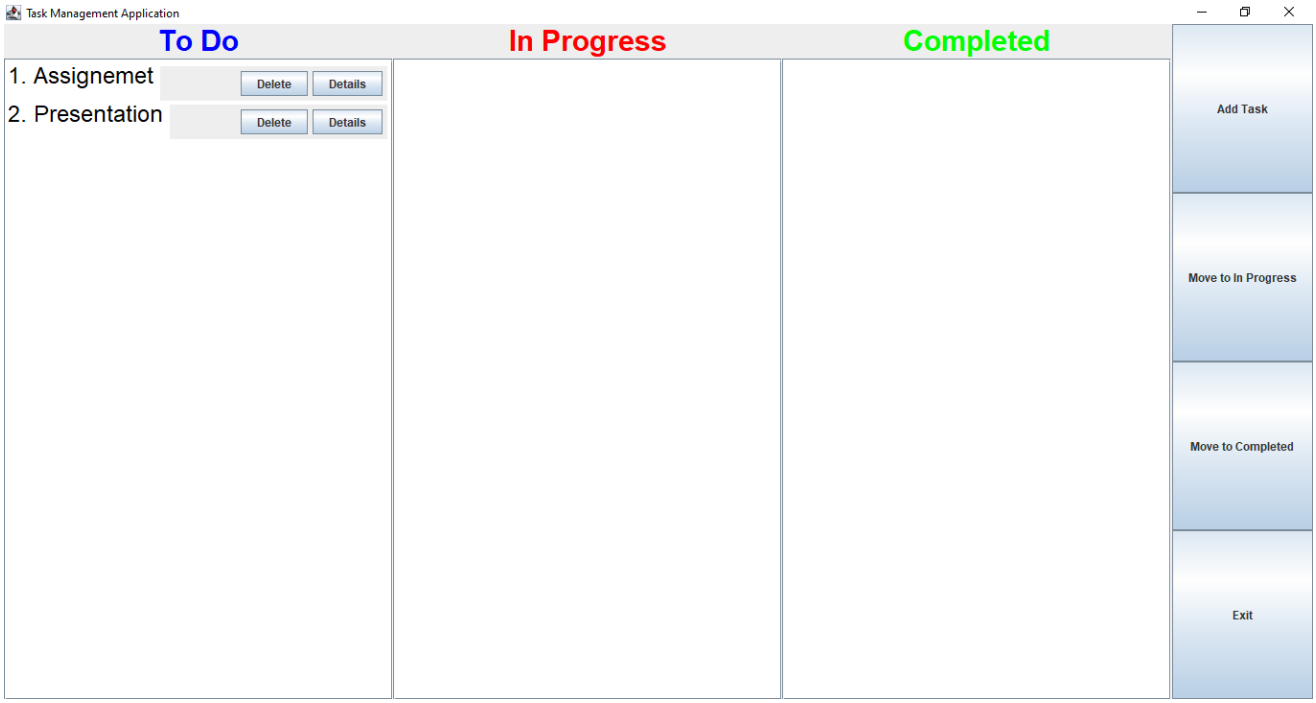
If you want to see the details of your task click the Details button to see your Task Completion time and also Task Duration time.



Step 5: Delete Task

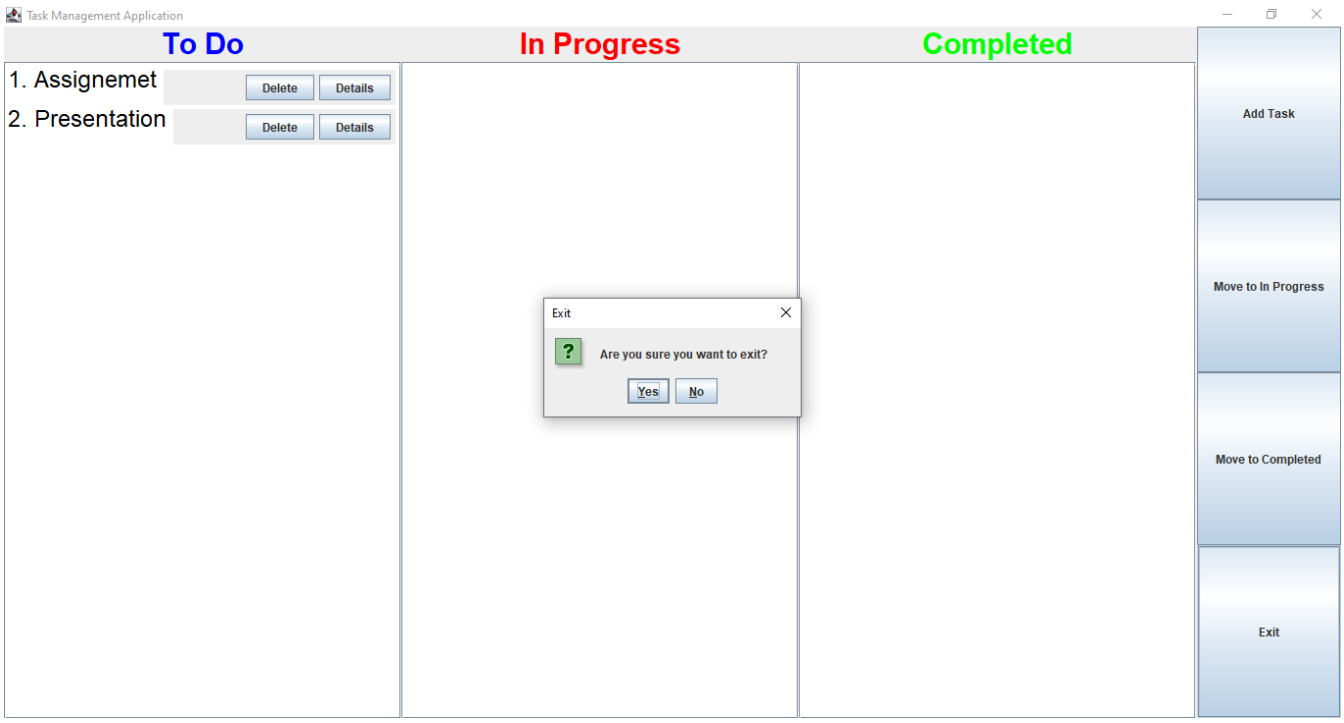
If you want to delete the task from any list click the delete button to simply delete the task.





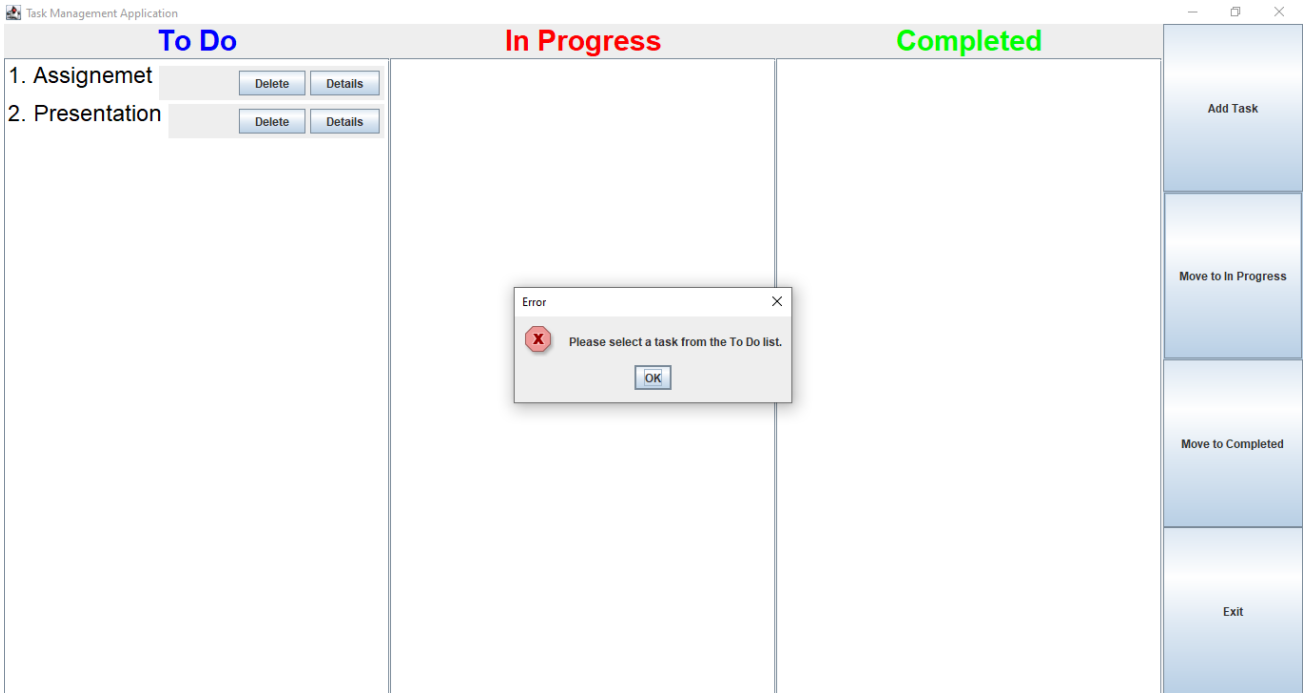
Step 6: Exit

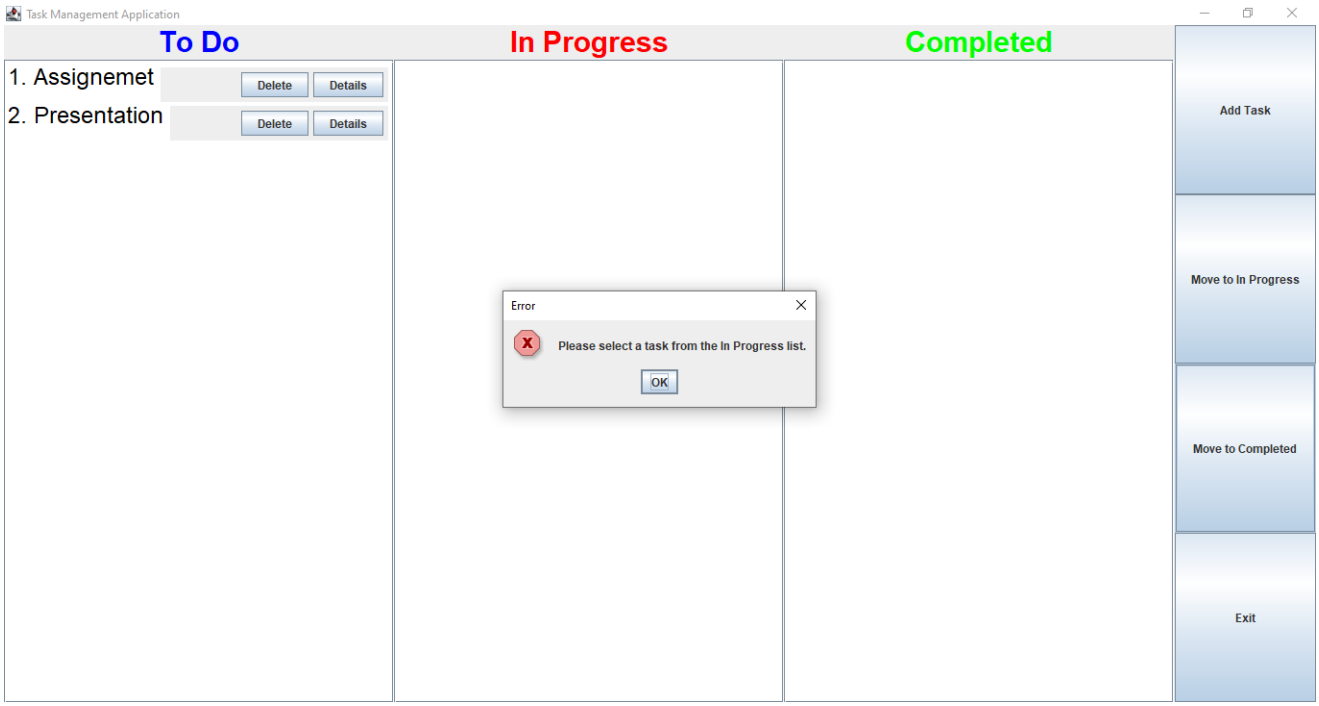
If you want to close the Task Management Application click the Exit button to close Application.



Others:

In the case of Wrong clicks





UNIT TESTING

Functionality 1

Test case Add Task (Valid)	
Test Case ID	1
Input	Description: "Complete assignment"
Partition tested	Valid Class
Expected Output	Task is added to the To-Do list
Actual Output	----------
Pass fail	----------

Test case Add Task (Invalid)	
Test Case ID	2
Input	Description: null
Partition tested	Invalid Class
Expected Output	Error message indicating invalid input
Actual Output	----------
Pass fail	----------

After Code Execution

Test case Add Task		
Test Case ID	1	2
Input	Description: "Complete assignment"	Description: null
Partition tested	Valid Class	Invalid Class
Expected Output	Task is added to the To-Do list	Error message indicating invalid input
Actual Output	Task is added to the To-Do list	Error message indicating invalid input
Pass fail	Pass	Pass

Functionality 2

Test case Task Detail (Valid)	
Test Case ID	3
Input	Click Detail Button in any List
Partition tested	Valid Class
Expected Output	Show Task Completion & Duration Time
Actual Output	----------
Pass fail	----------

Test case Task Detail (Invalid)	
Test Case ID	4
Input	Click Detail Button in any List
Partition tested	Invalid Class
Expected Output	Show Task not Completed
Actual Output	----------
Pass fail	----------

After Code Execution

Test case Task Deletion		
Test Case ID	3	4
Input	Click Detail Button in any List	Click Detail Button in any List
Partition tested	Valid Class	Invalid Class
Expected Output	Show Task Completion & Duration Time	Show Task not Completed
Actual Output	Show Task Completion & Duration Time	Show Task not Completed
Pass fail	Pass	Pass