**Name :NITHIN S**

**Reg no :231401073**

**OOPS WITH JAVA**

**ENHANCED TASK MANAGER**

**ENHANCED TASK MANAGER**

**Program:**

//TASK MANAGEMENT APPLICATION

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

// Enum for task priority

enum Priority {

HIGH, MEDIUM, LOW

}

public class EnhancedTaskManager {

private Frame mainFrame;

private TextField taskField;

private TextArea descriptionField;

private Choice priorityChoice;

private List taskList;

private Button submitButton, removeButton, markCompleteButton, clearAllButton, editButton, sortButton;

private Label notificationLabel;

private ArrayList<Task> tasks;

private ExecutorService executorService;

private Task editingTask = null;

// Track the task being edited

// Inner class to encapsulate task details

class Task {

String description;

Priority priority;

boolean completed;

Task(String description, Priority priority) {

this.description = description;

this.priority = priority;

this.completed = false;

}

@Override

public String toString() {

return (completed ? "[Completed] " : "") + description + " (" + priority + ")";

}

}

public EnhancedTaskManager() {

tasks = new ArrayList<>();

executorService = Executors.newFixedThreadPool(2); // Thread pool for task handling

prepareGUI();

}

private void prepareGUI() {

mainFrame = new Frame("Enhanced Task Manager");

mainFrame.setSize(500, 600);

mainFrame.setLayout(new FlowLayout(FlowLayout.LEFT));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent) {

executorService.shutdown();

System.exit(0);

}

});

Label taskLabel = new Label("Task:");

taskLabel.setFont(new Font("Arial", Font.BOLD, 14));

taskLabel.setForeground(Color.BLUE);

taskField = new TextField(35);

taskField.setFont(new Font("Arial", Font.PLAIN, 12));

Label descriptionLabel = new Label("Description:");

descriptionLabel.setFont(new Font("Arial", Font.BOLD, 14));

descriptionLabel.setForeground(Color.BLUE);

descriptionField = new TextArea(3, 35);

descriptionField.setFont(new Font("Arial", Font.PLAIN, 12));

Label priorityLabel = new Label("Priority:");

priorityLabel.setFont(new Font("Arial", Font.BOLD, 14));

priorityLabel.setForeground(Color.BLUE);

priorityChoice = new Choice();

priorityChoice.add("HIGH");

priorityChoice.add("MEDIUM");

priorityChoice.add("LOW");

taskList = new List();

taskList.setFont(new Font("Arial", Font.PLAIN, 12));

submitButton = new Button("Add Task");

submitButton.setBackground(Color.GREEN);

submitButton.setForeground(Color.WHITE);

submitButton.addActionListener(e -> executorService.execute(this::addOrEditTask));

removeButton = new Button("Remove Task");

removeButton.setBackground(Color.RED);

removeButton.setForeground(Color.WHITE);

removeButton.addActionListener(e -> executorService.execute(this::removeTask));

markCompleteButton = new Button("Mark Complete");

markCompleteButton.setBackground(Color.ORANGE);

markCompleteButton.setForeground(Color.WHITE);

markCompleteButton.addActionListener(e -> executorService.execute(this::markTaskComplete));

clearAllButton = new Button("Clear All Tasks");

clearAllButton.setBackground(Color.GRAY);

clearAllButton.setForeground(Color.WHITE);

clearAllButton.addActionListener(e -> executorService.execute(this::clearAllTasks));

editButton = new Button("Edit Task");

editButton.setBackground(Color.CYAN);

editButton.setForeground(Color.BLACK);

editButton.addActionListener(e -> executorService.execute(this::editTask));

sortButton = new Button("Sort Tasks");

sortButton.setBackground(Color.MAGENTA);

sortButton.setForeground(Color.WHITE);

sortButton.addActionListener(e -> executorService.execute(this::sortTasks));

notificationLabel = new Label("Welcome to the Task Manager!");

notificationLabel.setFont(new Font("Arial", Font.ITALIC, 12));

notificationLabel.setForeground(Color.DARK\_GRAY);

mainFrame.add(taskLabel);

mainFrame.add(taskField);

mainFrame.add(descriptionLabel);

mainFrame.add(descriptionField);

mainFrame.add(priorityLabel);

mainFrame.add(priorityChoice);

mainFrame.add(submitButton);

mainFrame.add(editButton);

mainFrame.add(removeButton);

mainFrame.add(markCompleteButton);

mainFrame.add(clearAllButton);

mainFrame.add(sortButton);

mainFrame.add(taskList);

mainFrame.add(notificationLabel);

mainFrame.setVisible(true);

}

private void addOrEditTask() {

try {

String description = taskField.getText().trim();

String details = descriptionField.getText().trim();

Priority priority = Priority.valueOf(priorityChoice.getSelectedItem());

if (description.isEmpty()) {

throw new IllegalArgumentException("Task cannot be empty.");

}

if (editingTask != null) {

editingTask.description = description + ": " + details;

editingTask.priority = priority;

editingTask = null; // Reset editing mode

notifyUser("Task edited successfully.");

} else {

Task newTask = new Task(description + ": " + details, priority);

tasks.add(newTask);

notifyUser("Task added successfully.");

}

updateTaskList();

clearInputFields();

} catch (IllegalArgumentException ex) {

notifyUser("Error: " + ex.getMessage());

}

}

private void removeTask() {

try {

int selectedIndex = taskList.getSelectedIndex();

if (selectedIndex < 0) {

throw new IndexOutOfBoundsException("No task selected to remove.");

}

tasks.remove(selectedIndex);

notifyUser("Task removed successfully.");

updateTaskList();

} catch (IndexOutOfBoundsException ex) {

notifyUser("Error: " + ex.getMessage());

}

}

private void removeTask() {

try {

int selectedIndex = taskList.getSelectedIndex();

if (selectedIndex < 0) {

throw new IndexOutOfBoundsException("No task selected to remove.");

}

tasks.remove(selectedIndex);

notifyUser("Task removed successfully.");

updateTaskList();

} catch (IndexOutOfBoundsException ex) {

notifyUser("Error: " + ex.getMessage());

}

}

private void markTaskComplete() {

try {

int selectedIndex = taskList.getSelectedIndex();

if (selectedIndex < 0) {

throw new IndexOutOfBoundsException("No task selected to mark complete.");

}

Task task = tasks.get(selectedIndex);

task.completed = true;

notifyUser("Task marked as complete.");

updateTaskList();

} catch (IndexOutOfBoundsException ex) {

notifyUser("Error: " + ex.getMessage());

}

}

private void clearAllTasks() {

tasks.clear();

notifyUser("All tasks cleared.");

updateTaskList();

}

private void editTask() {

try {

int selectedIndex = taskList.getSelectedIndex();

if (selectedIndex < 0) {

throw new IndexOutOfBoundsException("No task selected to edit.");

}

editingTask = tasks.get(selectedIndex);

taskField.setText(editingTask.description.split(":")[0]);

descriptionField.setText(editingTask.description.split(":")[1]);

priorityChoice.select(editingTask.priority.name());

notifyUser("Editing task...");

} catch (IndexOutOfBoundsException ex) {

notifyUser("Error: " + ex.getMessage());

}

}

private void sortTasks() {

tasks.sort(Comparator.comparing(task -> task.priority));

notifyUser("Tasks sorted by priority.");

updateTaskList();

}

private void updateTaskList() {

EventQueue.invokeLater(() -> {

taskList.removeAll();

for (Task task : tasks) {

taskList.add(task.toString());

}

});

}

private void clearInputFields() {

EventQueue.invokeLater(() -> {

taskField.setText("");

descriptionField.setText("");

priorityChoice.select("LOW");

});

}

private void notifyUser(String message) {

EventQueue.invokeLater(() -> notificationLabel.setText(message));

}

public static void main(String[] args) {

new EnhancedTaskManager();

}

}

**OUTPUT**:

