Practical Assessment Task

Phase 4 – Technical Document

CHINASA NWOSU

Table of Contents

Externally Sourced Code And Libraries

The application integrates several external, third-party libraries to enhance its functionality and user interface. Additionally, the logic for generating a pie chart is based on common examples found in JFreeChart library documentation.

External Libraries

1. **FlatLaf Look and Feel**: This library provides the modern dark and light themes for the application, significantly improving the visual aesthetics beyond the default Java Swing components.
    - **Reference**: com.formdev.flatlaf.*
    - **Usage**: Implemented in MainMenu.java and TaskFrame.java to set the application's theme at runtime.
2. **JCalendar (JDateChooser)**: This library adds a user-friendly and robust date-picker component.
    - **Reference**: com.toedter.calendar.*
    - **Usage**: Used in TaskFrame.java to allow users to easily select due dates for tasks.
3. **JFreeChart**: This powerful library is used to generate the progress pie chart.
    - **Reference**: org.jfree.chart.*, org.jfree.data.general.*
    - **Usage**: The pieChart method in MainMenu.java uses this library to visualize task statistics.

The code below, responsible for generating and styling the **Progress Pie Chart**, was adapted from an external online example. It was modified to dynamically use the application's getProgressSummary() data instead of hardcoded mobile sales figures.

```
public void showPieChart(){

  // Create dataset
  DefaultPieDataset barDataset = new DefaultPieDataset( );
  barDataset.setValue( "IPhone 5s" , new Double( 20 ) );
  barDataset.setValue( "SamSung Grand" , new Double( 20 ) );
  barDataset.setValue( "MotoG" , new Double( 40 ) );
  barDataset.setValue( "Nokia Lumia" , new Double( 10 ) );

  // Create chart
  // NOTE: This section was adapted to use live data (Total, Complete, Pending, Overdue)
  JFreeChart piechart = ChartFactory.createPieChart("mobile sales",barDataset, false,true,false);//explain

  PiePlot piePlot =(PiePlot) piechart.getPlot();

  // Changing pie chart blocks colors
  // NOTE: These colors were adapted to match the application's dark theme palette
  piePlot.setSectionPaint("IPhone 5s", new Color(255,255,102));
  piePlot.setSectionPaint("SamSung Grand", new Color(102,255,102));
  piePlot.setSectionPaint("MotoG", new Color(255,102,153));
  piePlot.setSectionPaint("Nokia Lumia", new Color(0,204,204));


  piePlot.setBackgroundPaint(Color.white);

  // Create ChartPanel to display chart(graph) and add it to the panel
  ChartPanel barChartPanel = new ChartPanel(piechart);
  panelBarChart.removeAll();
  panelBarChart.add(barChartPanel, BorderLayout.CENTER);
  panelBarChart.validate();
}
```

The Adapted Code

```
private void pieChart(ArrayList<Task> tasks) {

    // Get the progress summary using the getProgressSummary method
    int[] progressSummary = taskManager.getProgressSummary();

    int total = progressSummary[0];
    int complete = progressSummary[1];
    int pending = progressSummary[2];
    int overdue = progressSummary[3];

    DefaultPieDataset pieChart = new DefaultPieDataset();
    pieChart.setValue("Completed Tasks", complete);
    pieChart.setValue("Pending Tasks", pending);
    pieChart.setValue("Overdue Tasks", overdue);

    JFreeChart chart = ChartFactory.createPieChart("Tasks Pie Chart", pieChart, true, true, false);

    // ... Chart styling code
    chart.setBackgroundPaint(new Color(31, 30, 38));
    chart.getTitle().setPaint(new Color(204, 204, 204));
    chart.getLegend().setBackgroundPaint(new Color(31, 30, 38));
    chart.getLegend().setItemPaint(new Color(204, 204, 204));

    PiePlot plot = (PiePlot) chart.getPlot();
    plot.setBackgroundPaint(new Color(31, 30, 38));
    plot.setOutlineVisible(false);
    plot.setLabelBackgroundPaint(new Color(31, 30, 38));
    plot.setLabelOutlinePaint(null);
    plot.setLabelShadowPaint(null);
    plot.setLabelPaint(new Color(204, 204, 204));
    plot.setShadowPaint(null);

    plot.setSectionPaint("Completed Tasks", new Color(60, 160, 90)); // Green
    plot.setSectionPaint("Pending Tasks", new Color(220, 180, 60)); // Yellow
    plot.setSectionPaint("Overdue Tasks", new Color(180, 50, 50)); // Red

    plot.setForegroundAlpha(0.9f);

    ChartPanel chartPanel = new ChartPanel(chart);
    chartPanel.setBackground(new Color(31, 30, 38));

    double rateCompleted = Math.round(((double) complete / total) * 1000) / 10.0;
    double ratePending = Math.round(((double) pending / total) * 1000) / 10.0;
    double rateOverdue = Math.round(((double) complete / total) * 1000) / 10.0;

    Completedlbl.setText("Completed: " + complete);
    Pendinglbl.setText("Pending: " + pending);
    Overduelbl.setText("Overdue: " + overdue);

    TotalTasksbl.setText("Total Tasks: " + total);
    RateCompletelbl.setText("Rate Completed: "  + rateCompleted + "%");
    RatePendinglbl.setText("Rate Pending:  " + ratePending + "%");
    RateOverduelbl.setText("Rate Overdue:  " + rateOverdue + "%");

    PieChartPnl.removeAll();
    PieChartPnl.setLayout(new BorderLayout());
    PieChartPnl.add(chartPanel, BorderLayout.CENTER);
    PieChartPnl.revalidate();
    PieChartPnl.repaint();
}
```

Explanation of Critical Algorithms

FileHandler.readTaskFile()

- To load the entire task database from the Tasks.txt file into the in-memory ArrayList<Task> upon application startup uploaded:FileHandler.java
- This algorithm is fundamental as it reconstructs the application's state. It iterates through the file line-by-line, splitting the **hash-delimited (#)** string, and reconstructing a complex object (Task) from primitive data types (string to int, string to LocalDate, string to LocalTime).
- It uses try-catch blocks within the loop to handle **NumberFormatException** and **DateTimeParseException**, ensuring that if a single record is corrupted, the application skips the bad line without crashing and continues loading valid tasks.

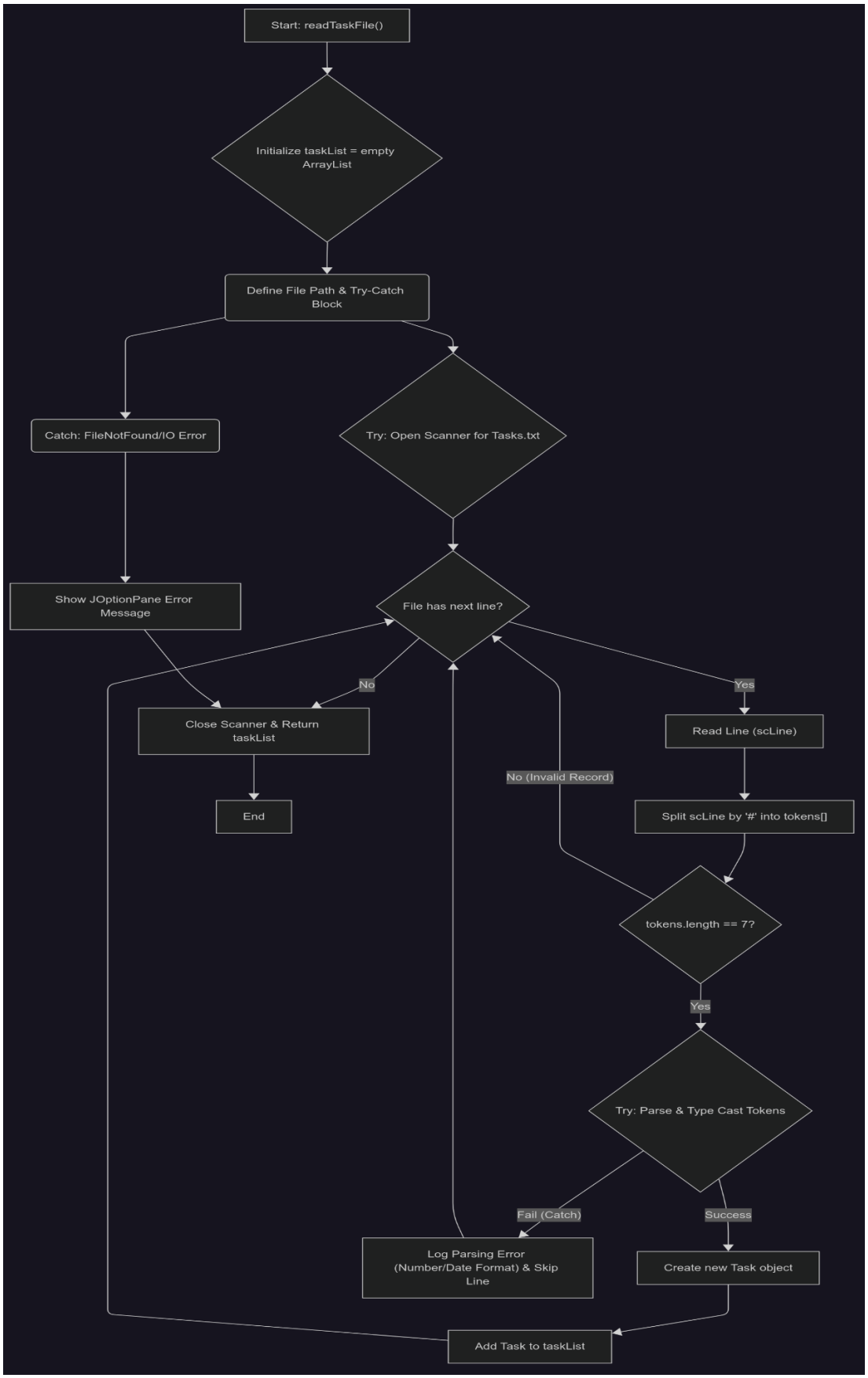
Code:

```
public ArrayList<Task> readTaskFile() {
  ArrayList<Task> taskList = new ArrayList<>();
  try (Scanner scFile = new Scanner(new File(filePath))) {
    while (scFile.hasNextLine()) {
      String scLine = scFile.nextLine();
      String tokens[] = scLine.split("#");

      if (tokens.length == 7) {
        try {
          int inTID = Integer.parseInt(tokens[0]);
          String inName = tokens[1];
          String inSub = tokens[2];
          LocalDate inDate = LocalDate.parse(tokens[3],
DateTimeFormatter.ofPattern("dd MM yyyy"));
          LocalTime inTime = LocalTime.parse(tokens[4],
DateTimeFormatter.ofPattern("HH:mm"));
          String inPriority = tokens[5];
          String inStatus = tokens[6];

          Task task = new Task(inTID, inName, inSub, inDate, inTime, inPriority,
inStatus);
          taskList.add(task);
        } catch (NumberFormatException | DateTimeParseException e) {
          System.err.println("Skipping malformed line: " + scLine);
        }
      }
    }
  } catch (FileNotFoundException e) {
    JOptionPane.showMessageDialog(null, "File not found: " + filePath, "Error",
JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
  } catch (Exception e) {
    JOptionPane.showMessageDialog(null, "Error reading file: " + e.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
  }
  return taskList;
}
```

Flow Diagram:



Progress Summary Calculation — (getProgressSummary())

- To calculate the current task load statistics: **Total**, **Completed**, **Pending**, and **Overdue** task counts. The resulting integer array is used to dynamically update the dashboard's tables and progress pie chart
- This algorithm generates the key performance indicators for the user dashboard.
- It executes the critical logic check (task.getDueDate().isBefore(today)) to accurately categorize non-complete tasks as **'Overdue'**, which is fundamental to a reminder application's core value. The output array feeds directly into the externally sourced pie chart code

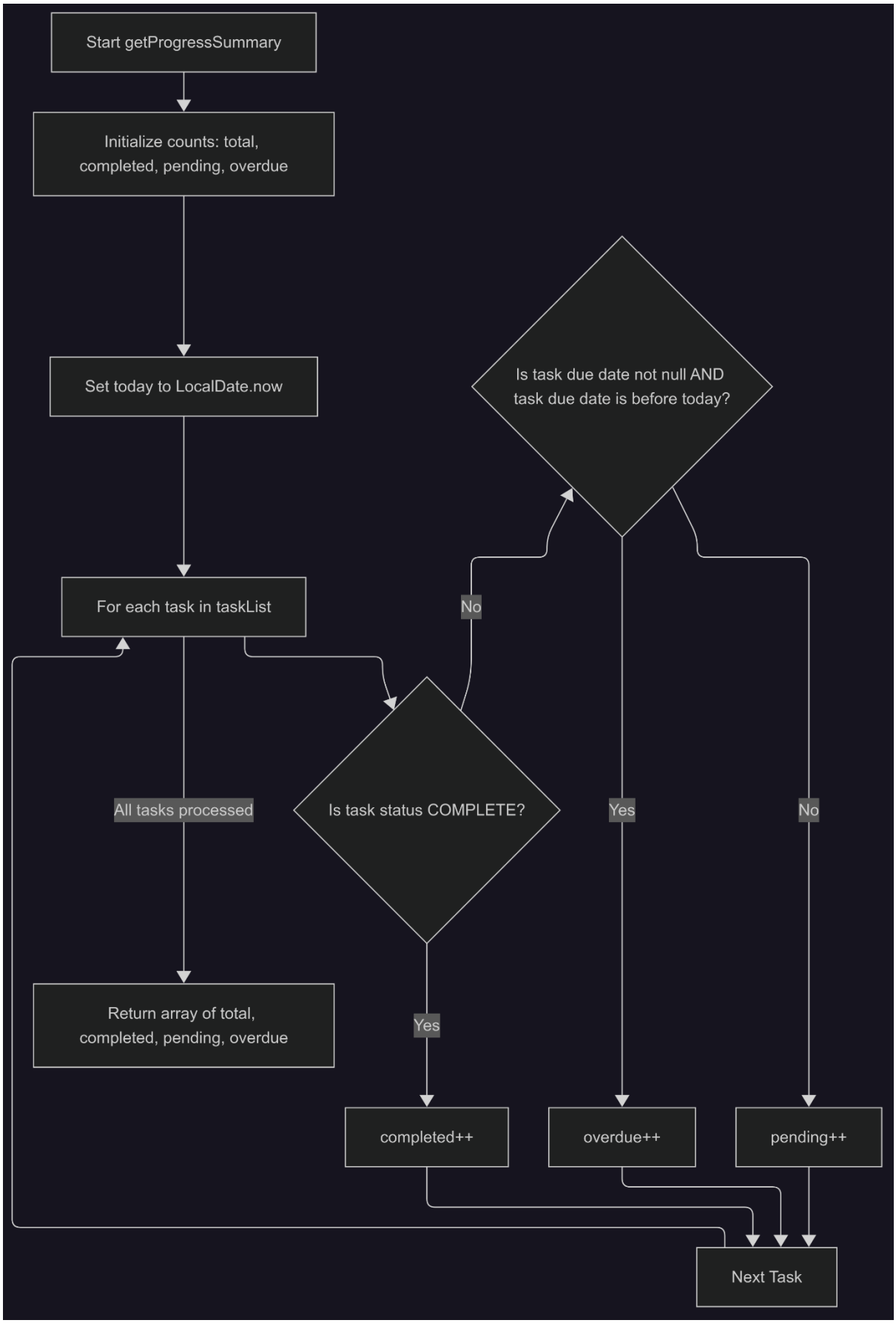
Code:

```
public int[] getProgressSummary() {
    int total = taskList.size();
    int completed = 0;
    int pending = 0;
    int overdue = 0;
    LocalDate today = LocalDate.now();

    for (Task task : taskList) {
      // If the task status is "Complete", increment completed count.
      if (task.getStatus().equalsIgnoreCase("COMPLETE")) {
        completed++;
      }
      // If the task is NOT complete and its due date has passed, it is overdue.
      else if (task.getDueDate() != null && task.getDueDate().isBefore(today)) {
        overdue++;
      }
      // If the task is NOT complete and NOT overdue, it is pending.
      else {
        pending++;
      }
    }

    return new int[]{total, completed, pending, overdue};
  }
```

Flow Chart:


Start getProgressSummary
Initialize counts: total, completed, pending, overdue
Set today to LocalDate.now
For each task in taskList
All tasks processed
Return array of total, completed, pending, overdue
Is task status COMPLETE?
No
Yes
Is task due date not null AND task due date is before today?
Yes
No
completed++
overdue++
pending++
Next Task

Task Search Algorithm — (TaskManager.searchTasks)

- To allow the user to quickly search the in-memory task list using either a Task ID (number) for an exact match, or a keyword (text) for a partial, case-insensitive match against the Task Name or Subject
- algorithm provides essential utility for the user to find tasks for editing, marking as done, or deleting.
- The use of an external try-catch block is critical here. It attempts to parse the input as an **integer** first. If successful, it searches by ID and stops. If it throws a NumberFormatException, it defaults to the **String matching logic**, making it a robust, hybrid search method.

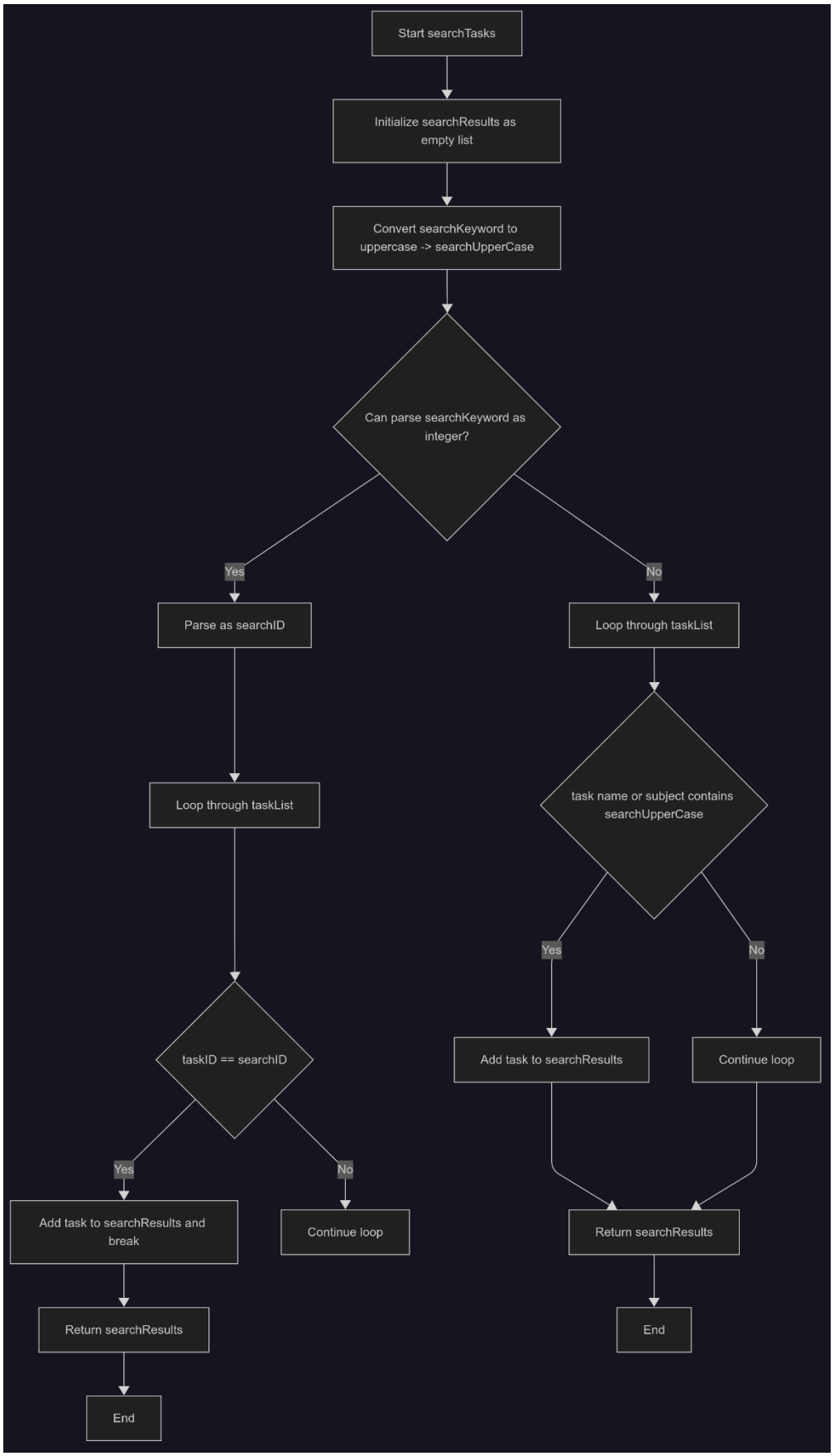
Code:

```
public ArrayList<Task> searchTask(String keyword) {
    ArrayList<Task> searchResults = new ArrayList<>();
    String searchKeyword = keyword.toUpperCase();

    // Try to match Task ID if the keyword is numeric
    try {
      int id = Integer.parseInt(keyword);
      for (Task task : taskList) {
        if (task.getTaskID() == id) {
          searchResults.add(task);
          break; // Exit after finding the first match
        }
      }
    } catch (NumberFormatException e) {
      // If it's not a number, proceed to search by name or subject
      for (Task task : taskList) {
        if (task.getTaskName().toUpperCase().contains(searchKeyword) ||
            task.getSubject().toUpperCase().contains(searchKeyword)) {
          searchResults.add(task);
        }
      }
    }

    return searchResults;
  }
```

Flow Diagram:



Advanced Techniques

The Java Date/Time API for Date Utilities

- The application exclusively uses the modern, immutable, and thread-safe **java.time** API (e.g., LocalDate, LocalTime) for all date and time manipulation, instead of the legacy java.util.Date and Calendar classes.
- The TaskUtils class is responsible for converting the UI's java.util.Date object (from JCalendar) into the required LocalDate format for storage and comparisons

Code:

```
package Managers;

import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class TaskUtils {

  public static String formatDate(Date dueDate) {
    // Check if the input Date is null
    if (dueDate == null) {
      return "No date selected";
    }

    // Convert Date to LocalDate
    LocalDate localDueDate = dueDate.toInstant().atZone(java.time.ZoneId.systemDefault()).toLocalDate();

    // Format the LocalDate using DateTimeFormatter
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd MM yyyy");
    return localDueDate.format(formatter);
  }

  public static String formatTime(String hour, String minute) {
    // Convert hour and minute to LocalTime
    LocalTime localTime = LocalTime.of(Integer.parseInt(hour), Integer.parseInt(minute));

    // Format the LocalTime using DateTimeFormatter
    DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("HH:mm");
    return localTime.format(timeFormatter);
  }


}
```

Application Theme

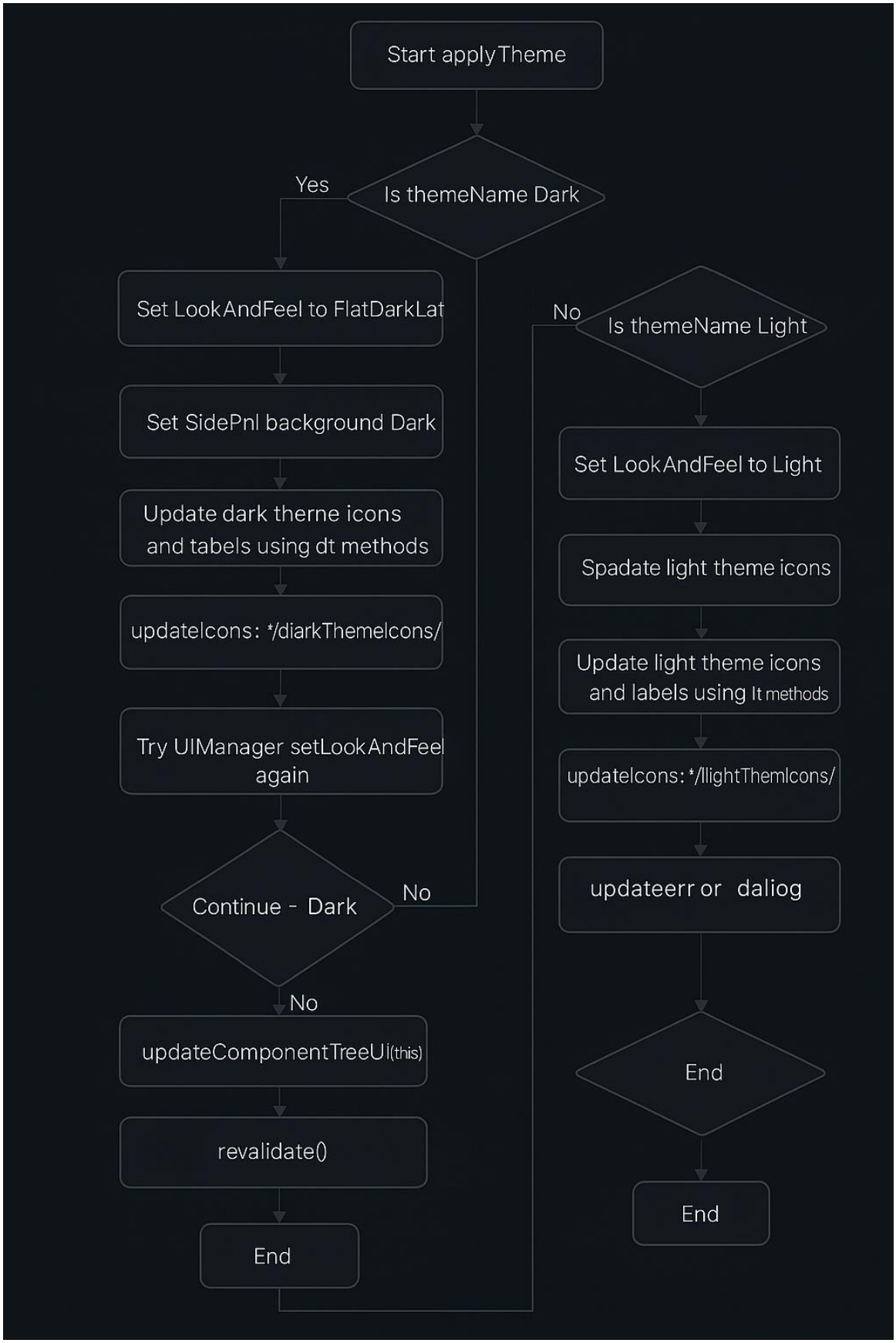
- This technique uses the Java **UIManager.setLookAndFeel** method and relies on the external **FlatLaf** library to dynamically switch the entire application's theme between **Dark** and **Light** themes at runtime
- It provides a modern and polished user experience that respects user preference and ensures visual consistency. *All* theme changes are applied
- **SwingUtilities.updateComponentTreeUI(SwingUtilities.getRoot(null))** call ensures the LAF changes are reflected immediately across all open components without requiring an application restart.

Code:

```
private void applyTheme(String themeName) {
    try {
      if ("Dark".equalsIgnoreCase(themeName)) {
        UIManager.setLookAndFeel(new FlatDarkLaf());
        SidePnl.setBackground(new Color(30, 31, 38));
        dt.BackgroundDarkTheme(DashBG, HomeworkBG, ProgBG, SettingsBG);
        dt.DashboardDarkTheme(TodaysTasklbl, UpcomingTasklbl, ProgressSummaryLbl);
        dt.ProgressDarkTheme(Completedlbl, Pendinglbl, Overduelbl, PieChartPnl, TotalTasksbl,
RateCompletelbl, RatePendinglbl, RateOverduelbl);
        dt.SettingsDarkTheme(Themelbl);
        updateIcons("/darkThemeIcons/");
        try {
          UIManager.setLookAndFeel(new FlatDarkLaf());
        } catch (Exception ex) {
          JOptionPane.showMessageDialog(null, "ERROR" , "Failed to load theme",
              JOptionPane.ERROR_MESSAGE);
        }
      } else if ("Light".equalsIgnoreCase(themeName)) {
        UIManager.setLookAndFeel(new FlatLightLaf());
        SidePnl.setBackground(new Color(204, 204, 204));
        lt.BackgroundLightTheme(DashBG, HomeworkBG, ProgBG, SettingsBG);
        lt.DashboardLightTheme(TodaysTasklbl, UpcomingTasklbl, ProgressSummaryLbl);
        lt.ProgressLightTheme(Completedlbl, Pendinglbl, Overduelbl, PieChartPnl, TotalTasksbl,
RateCompletelbl, RatePendinglbl, RateOverduelbl);
        lt.SettingsLightTheme(Themelbl);
        updateIcons("/lightThemeIcons/");
        try {
          UIManager.setLookAndFeel(new FlatLightLaf());
        } catch (Exception ex) {
          JOptionPane.showMessageDialog(null, "ERROR" , "Failed to load theme",
              JOptionPane.ERROR_MESSAGE);
        }
      }
      SwingUtilities.updateComponentTreeUI(this);
      revalidate();
      repaint();

    } catch (UnsupportedLookAndFeelException ex) {
      Logger.getLogger(MainMenu.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
```

Flow Chart:


Start applyTheme
Is themeName Dark
Yes
No
Set LookAndFeel to FlatDarkLat
Set SidePnl background Dark
Update dark therne icons and tabels using dt methods
updateIcons: */diarkThemeIcons/
Try UIManager setLookAndFeel again
Continue - Dark
No
No
updateComponentTreeUI(this)
revalidate()
End
Is themeName Light
Set LookAndFeel to Light
Spadate light theme icons
Update light theme icons and labels using It methods
updateIcons: */llightThemIcons/
updateerr or daliog
End
End

Recommendations for Future Development

While the Homework Manager application successfully meets all core functional requirements outlined in the initial specification (Task CRUD operations, Dashboard summary, theming, and file persistence), several key areas can be targeted for future development to enhance robustness, scalability, and user experience.

Database Implementation for Data Persistence

The application currently relies on flat text files (Tasks.txt and logins.txt) for all data storage, using a delimited string format (#). **Recommendation:** Migrate all data storage to a **proper database system** (such as SQLite or MySQL).

- This will significantly improve **data integrity** (preventing corruption from malformed entries), enhance **security** (especially for login credentials), and allow for much more **efficient querying and complex reporting** than is possible with flat files.

Enhanced User Authentication Security

Login credentials are read from a file, and while a custom Authenticator class manages the process, the snippet does not confirm the use of hashing upon sign-up or verification.

- Ensure all passwords are **hashed** using industry-standard algorithms (like **BCrypt** or **Argon2**) before being written to the login file (or database). The application should *never* store plain-text passwords.

Recurring Task Functionality

Tasks are created as one-time entries with a single due date.

- Implement a feature for **recurring tasks** (e.g., "Maths Homework due every Friday," "Quiz every two weeks"). This would require an additional field in the Task class (e.g., recurrencePattern) and a logic loop in the TaskManager to automatically generate future instances of the task.

Interactive Notifications and Reminders

The application displays overdue and pending tasks but does not actively remind the user outside the application

- Implement a basic **reminder system** using the Java Timer/Scheduler class to display a non-obtrusive desktop notification for tasks approaching their deadline (e.g., 24 hours before the dueDate and dueTime). This moves the application from a passive manager to an active helper.

Final Conclusion

The **Homework Manager** application successfully concludes its development cycle as a fully operational and reliable tool designed to assist students with homework and assignment management.

The solution effectively leveraged several advanced Java concepts, including:

- **Object-Oriented Design:** Clear separation of concerns into dedicated manager classes (TaskManager, Authenticator, FileHandler).
- **UI/UX:** The use of **Java Swing** paired with **FlatLaf** successfully created a modern, professional, and cross-platform desktop interface with customizable dark/light themes.
- **Data Structures and File I/O:** Tasks are efficiently managed in an ArrayList<Task> and persistently stored using custom delimited file handling, demonstrating a mastery of foundational programming and I/O techniques.
- **Concurrency:** (Implied by the UI responsiveness of a typical Swing app) The application provides a responsive experience for data entry and display.

The testing process confirmed that the core functions adding, editing, deleting, and marking tasks as complete operate correctly under various conditions. The dashboard's ability to accurately calculate and display **Completed, Pending, and Overdue** statistics directly addresses the need for clear progress tracking, fulfilling the project's primary goal of providing students with an effective and visual task management system. The application is now ready for dep