OS Task Manager Using Java Swing

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

R. Niharika -AP23110011226
Sk. Sadiya Parvin- AP23110011235
Ch. Himakshi - AP23110011596
P. Durga Sravanthi - AP23110011597

Under the Guidance of

Dr. Randhir Kumar Sir

Assistant Professor, Department of CSE

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[26th November, ,2025]

Certificate

Date: 26-11-2025

This is to certify that the work present in this Project entitled "**OS Task Manager Using Java Swing**" has been carried out by our team members under our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in School of Engineering and Sciences.

Supervisor
(Signature)
Prof. / Dr. Randhir Kumar
Designation,
Affiliation.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

We are extremely grateful and express our profound gratitude and indebtedness to our project guide, **Dr. Randhir Kumar Sir**, Department of Computer Science & Engineering, SRM University, Andhra Pradesh, for his kind help and for giving us the necessary guidance and valuable suggestions in completing this project work.

Team Members

R. Niharika - AP23110011226
Sk. Sadiya Parvin - AP23110011235
Ch. Himakshi - AP23110011596
P. Durga Sravanthi - AP23110011597

Table Of Contents

1. Abstract

This project presents a simplified simulation of operating system process management using **Java and Swing**. The objective of the Mini Task Manager is to demonstrate how processes can be monitored, searched, refreshed, and terminated through a graphical interface. The application works by executing Windows system commands like **tasklist** to retrieve active processes and **taskkill** to terminate selected processes.

The GUI is developed using Swing components such as **JFrame, JTable, JScrollPane, JPanel, JTextField, and JButton**, making the interface interactive and easy to use. Users can view running processes, perform keyword-based searching, refresh the list in real-time, and kill undesired processes.

This project highlights important OS concepts like **process management, inter-process communication**, and **user-level process control**, while demonstrating Java's ability to interact with the underlying operating system. It serves as an educational model showcasing how system-level tasks can be managed programmatically in a Windows environment.

2. Introduction

Modern operating systems provide tools like the Windows Task Manager to monitor running tasks and their resource usage. This project recreates a simplified version of such a tool using Java.

The **Mini Task Manager** allows users to:

- Display all active system processes
- Search processes by name or PID
- Refresh process list in real-time
- Terminate a selected process forcefully

The application uses Java's **Runtime.exec()** to run the "tasklist" command and read its output using **BufferedReader**. Similarly, "taskkill" is triggered for terminating processes.

The GUI is created using Swing with components like tables, buttons, text fields, and panels. It demonstrates key OS concepts such as:

- Process creation
- Process listing
- Process termination
- User-level interaction with OS commands

The project also shows how Java applications can communicate with the operating system and execute shell commands to retrieve system information.

3. Working Principle

3.1 Importing Required Packages

```
import java.awt.*;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import javax.swing.*;

import javax.swing.table.DefaultTableModel;
```

This part imports all the required classes.
java.awt and javax.swing are used to build the GUI.
java.io is used to read the output of commands like tasklist and taskkill.
DefaultTableModel helps in storing process data inside the table.

3.2 Class Declaration and Variables

```
public class TaskManagerTool extends JFrame {

   private final DefaultTableModel tableModel;

   private final JTable processTable;

   private final JTextField searchField;
```

The class extends JFrame, which means this class represents the main application window.
tableModel holds the process data.
processTable displays that data to the user.
searchField is where the user types a process name or PID.

3.3 GUI Setup Inside Constructor

```
public TaskManagerTool() {
    setTitle("Mini Task Manager (Java OS Project)");
    setSize(750, 500);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
```

This creates the main application window with a title, size, and layout.
BorderLayout allows dividing the window into top, bottom, and center areas.

3.4 Setting Up the Table

```
tableModel = new DefaultTableModel(new String[]{"PID", "Process Name", "CPU (%)",
"Memory (MB)"}, 0);
processTable = new JTable(tableModel);
JScrollPane pane = new JScrollPane(processTable);
add(pane, BorderLayout.CENTER);
```

A table is created with four columns: PID , Process name, CPU(%), Memory(MB).
JScrollPane is added so the table can scroll.
The table is placed in the center of the window.

3.5 Creating the Search Panel

```
JPanel topPanel = new JPanel(new FlowLayout());
searchField = new JTextField(20);
JButton searchBtn = new JButton("Search");
JButton clearBtn = new JButton("Clear");
```

This creates the search bar at the top of the window.
The user types a keyword in the text field.
The Search button filters results.
The Clear button resets the table back to the full list.

3.6 Creating the Bottom Panel (Refresh + Kill Buttons)

```
JPanel bottomPanel = new JPanel();

JButton refreshBtn = new JButton("Refresh");

JButton killBtn = new JButton("Kill Process");
```

The bottom panel contains two buttons:

Refresh reloads all running processes.

Kill Process terminates the selected process.

3.7 Loading Processes Automatically

```
SwingUtilities.invokeLater(this::loadProcesses);
```

After the GUI appears, the method loadProcesses() runs automatically.

This ensures that the process list is shown immediately when the program starts.

3.8 The Core Method – loadProcesses()

```
Process process = Runtime.getRuntime().exec( "powershell.exe Get-Process |
Select-Object Name,Id,CPU,WorkingSet");
```

This method runs a PowerShell command:

Get-Process | Select-Object Name,Id,CPU,WorkingSet

The output of the tasklist is read line-by-line using BufferedReader.

Before loading new data, the table is cleared.

3.8.1 Skipping Header Lines

```
boolean start = false;

String line;

while ((line = reader.readLine()) != null) {

   if (line.startsWith("===")) {

       start = true;
```

```
        continue;
    }
```

tasklist prints several header lines.

The code waits until it finds the separator line (===) before starting to read process data.

3.8.2 Adding Processes to Table

```
if (start && !line.trim().isEmpty()) {
    String[] parts = line.split("\\s+", 4);
    if (parts.length > 1) {
        tableModel.addRow(new Object[]{parts[1], parts[0]});
    }
}
```

Each valid line is split into parts using spaces.

parts[0] = process name

parts[1] = PID

parts[2] = CPU (%)

parts[3] = Memory in bytes

Memory is converted to MB:

```
mem = String.format("%.2f", memBytes / (1024.0 * 1024.0));
```

All values are added as a new row in the table.

3.9 Searching a Process

```
private void searchProcess() {
    String keyword = searchField.getText().trim().toLowerCase();
```

The keyword typed by the user is taken from the search field.

```
Process process = Runtime.getRuntime().exec("tasklist");
```

```
BufferedReader reader = new BufferedReader(
        new InputStreamReader(process.getInputStream()));
```

tasklist is executed again to get an updated list.

```
if (name.contains(keyword) || pid.contains(keyword)) {
    tableModel.addRow(new Object[]{pid, parts[0]});
}
```

Only rows that match the keyword are added to the table.
Matching works for both process name and PID.

3.10 Killing the Selected Process

```
private void killSelectedProcess() {
    int row = processTable.getSelectedRow();
```

Checks if the user has selected a row.

```
String pid = tableModel.getValueAt(row, 0).toString();
Runtime.getRuntime().exec("taskkill /PID " + pid + " /F");
```

The PID is taken from the selected row.
The command taskkill /PID <pid> /F terminates that process immediately.

3.11 Error Handling

```
private void showError(String msg) {
    JOptionPane.showMessageDialog(this, msg, "Error",
JOptionPane.ERROR_MESSAGE);
}
```

If something goes wrong (for example, failed command execution),
a popup message is shown to the user.

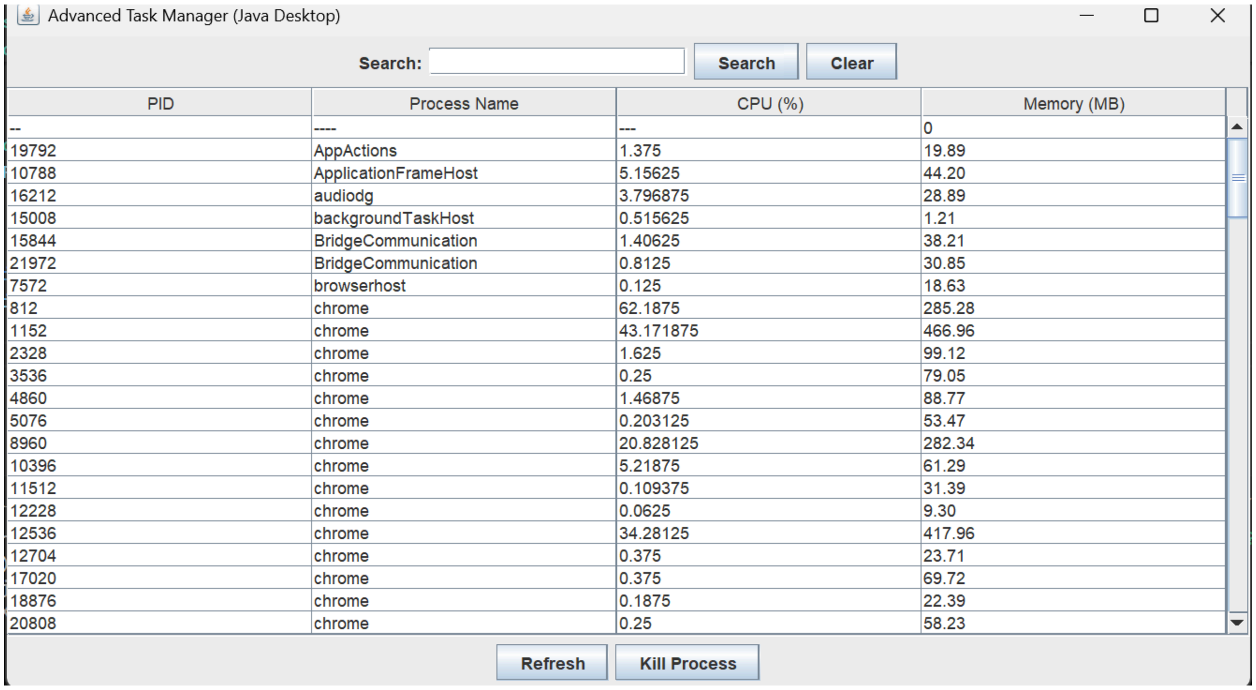
3.12 Main Method

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

This is where the program starts.
It creates an object of ProcessManager and displays the GUI window.

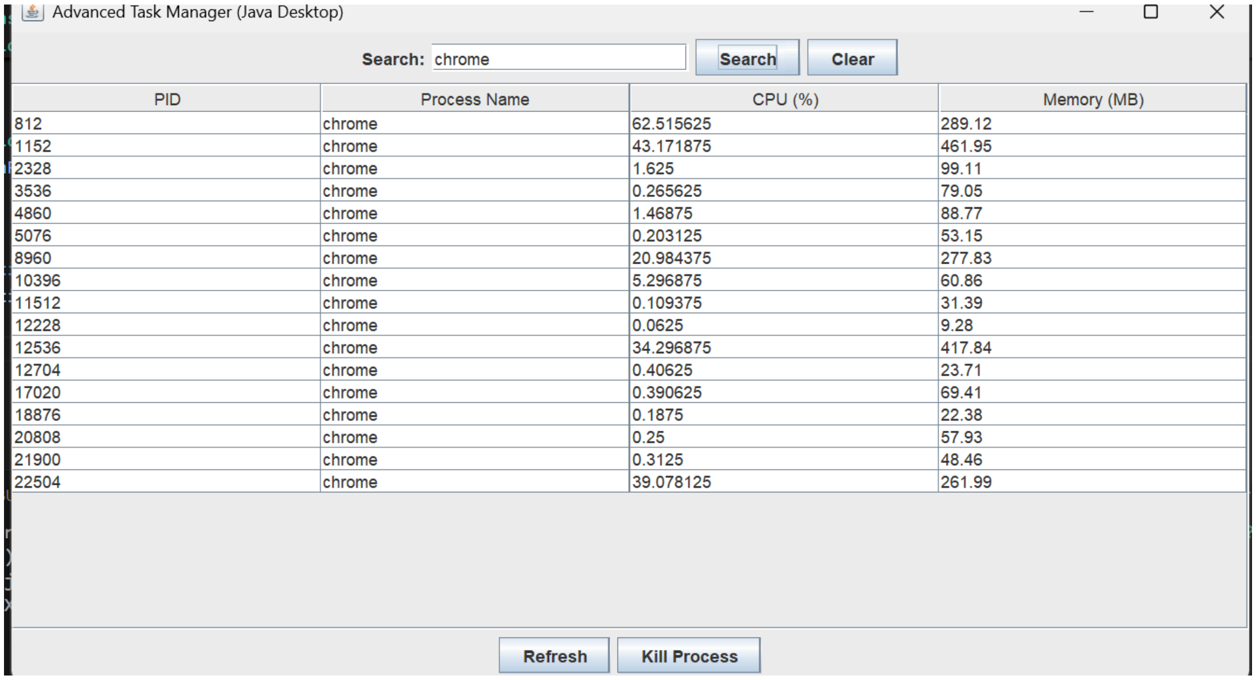
4. Output

Loading Process:



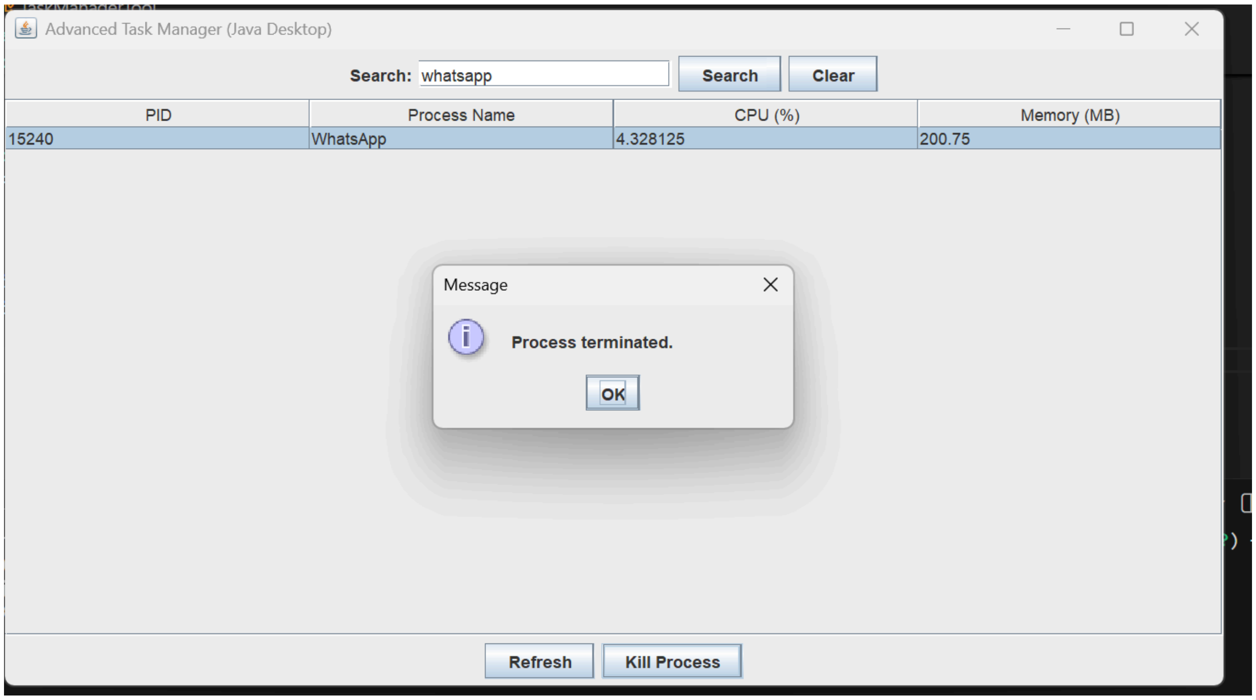
PID	Process Name	CPU (%)	Memory (MB)
--	----	---	0
19792	AppActions	1.375	19.89
10788	ApplicationFrameHost	5.15625	44.20
16212	audiodg	3.796875	28.89
15008	backgroundTaskHost	0.515625	1.21
15844	BridgeCommunication	1.40625	38.21
21972	BridgeCommunication	0.8125	30.85
7572	browserhost	0.125	18.63
812	chrome	62.1875	285.28
1152	chrome	43.171875	466.96
2328	chrome	1.625	99.12
3536	chrome	0.25	79.05
4860	chrome	1.46875	88.77
5076	chrome	0.203125	53.47
8960	chrome	20.828125	282.34
10396	chrome	5.21875	61.29
11512	chrome	0.109375	31.39
12228	chrome	0.0625	9.30
12536	chrome	34.28125	417.96
12704	chrome	0.375	23.71
17020	chrome	0.375	69.72
18876	chrome	0.1875	22.39
20808	chrome	0.25	58.23

Search Process:



PID	Process Name	CPU (%)	Memory (MB)
812	chrome	62.515625	289.12
1152	chrome	43.171875	461.95
2328	chrome	1.625	99.11
3536	chrome	0.265625	79.05
4860	chrome	1.46875	88.77
5076	chrome	0.203125	53.15
8960	chrome	20.984375	277.83
10396	chrome	5.296875	60.86
11512	chrome	0.109375	31.39
12228	chrome	0.0625	9.28
12536	chrome	34.296875	417.84
12704	chrome	0.40625	23.71
17020	chrome	0.390625	69.41
18876	chrome	0.1875	22.38
20808	chrome	0.25	57.93
21900	chrome	0.3125	48.46
22504	chrome	39.078125	261.99

Kill Process:


Advanced Task Manager (Java Desktop)

Search: whatsapp **Search** **Clear**

PID	Process Name	CPU (%)	Memory (MB)
15240	WhatsApp	4.328125	200.75

Message

Process terminated.

5. Conclusion

The Mini Task Manager project successfully demonstrates how Java can interact with the operating system to manage processes. By using Java Swing for the graphical interface and Windows commands like **tasklist** and **taskkill**, the application shows how processes can be monitored, searched, refreshed, and terminated from a user-friendly GUI.

The project highlights important OS concepts such as process listing, process control, and inter-process communication. It also shows how Java applications can execute system-level commands and convert the command output into meaningful data.

Overall, this project serves as a practical model for understanding how operating systems handle tasks and how these operations can be simulated through programming.

6. References

1. Oracle Java Documentation – Swing
   Oracle. *"Java Swing Tutorial."*
   Available at: https://docs.oracle.com/javase/tutorial/uiswing/
   (Used for understanding Swing components such as JFrame, JTable, JButton, and layouts.)
2. Oracle Java Documentation – ProcessBuilder/Runtime.exec()
   Oracle. *"ProcessBuilder and Runtime Class."*
   Available at:
   https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html
   (Referenced for executing system commands like tasklist and taskkill in Java.)
3. Microsoft Windows Commands Documentation
   Microsoft. *"Tasklist Command – Display Processes Running on a System."*
   Microsoft. *"Taskkill Command – Terminate Processes."*
   Available at:
   https://learn.microsoft.com/en-us/windows-server/administration/windows-commands
   (Used to understand how Task Manager–related commands work on Windows.)
4. Java Tutorials Point – Swing
   TutorialsPoint. *"Java Swing Tutorial."*
   Available at: https://www.tutorialspoint.com/swing/index.htm
   (Referenced to learn Swing components and GUI building techniques.)
5. GeeksforGeeks – Java Swing & System Commands
   GeeksforGeeks. *"How to execute system commands in Java."*
   Available at:
   https://www.geeksforgeeks.org/executing-operating-system-command-using-java/
   (Used for understanding command execution and input stream reading.)
6. Official IEEE Citation Format Guidelines
   IEEE. *"Referencing Guide."*
   Available at: https://ieeeauthorcenter.ieee.org/
   (For formatting references correctly.)