Advanced Task Manager

A Java Desktop Application for Windows Process Monitoring

A comprehensive system utility built with Java Swing that provides real-time process management capabilities on Windows, combining elegant GUI design with powerful system-level operations.

Submitted by:

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

Project Overview

Core Functionality

The Advanced Task Manager is a desktop-based system monitoring tool that displays real-time information about all running processes in Windows. It provides an intuitive graphical interface for viewing, searching, refreshing, and terminating system processes efficiently.

The application fetches process details using Windows PowerShell's Get-Process command, retrieving process name, PID, CPU usage, and memory consumption in a structured table format.

Key Technical Features

Real-Time Monitoring

Displays live process data including PID, process name, CPU percentage, and memory consumption in megabytes.

Smart Search

Filters processes dynamically by name or PID, enabling quick identification of specific applications.

Instant Refresh

Updates the process list on-demand to reflect current system state with a single click.

Process Termination

Forcefully kills selected processes using Windows taskkill command for complete control.

Application Architecture

01

GUI Initialisation

JFrame setup with BorderLayout, table model creation, and component positioning using Swing containers.

02

Process Data Retrieval

PowerShell command execution via Runtime.getRuntime().exec() to fetch process information from Windows.

03

Data Parsing

BufferedReader processes command output, extracting and formatting process details into table rows.

04

User Interaction

Event listeners handle button clicks for refresh, search, clear, and process termination actions.

05

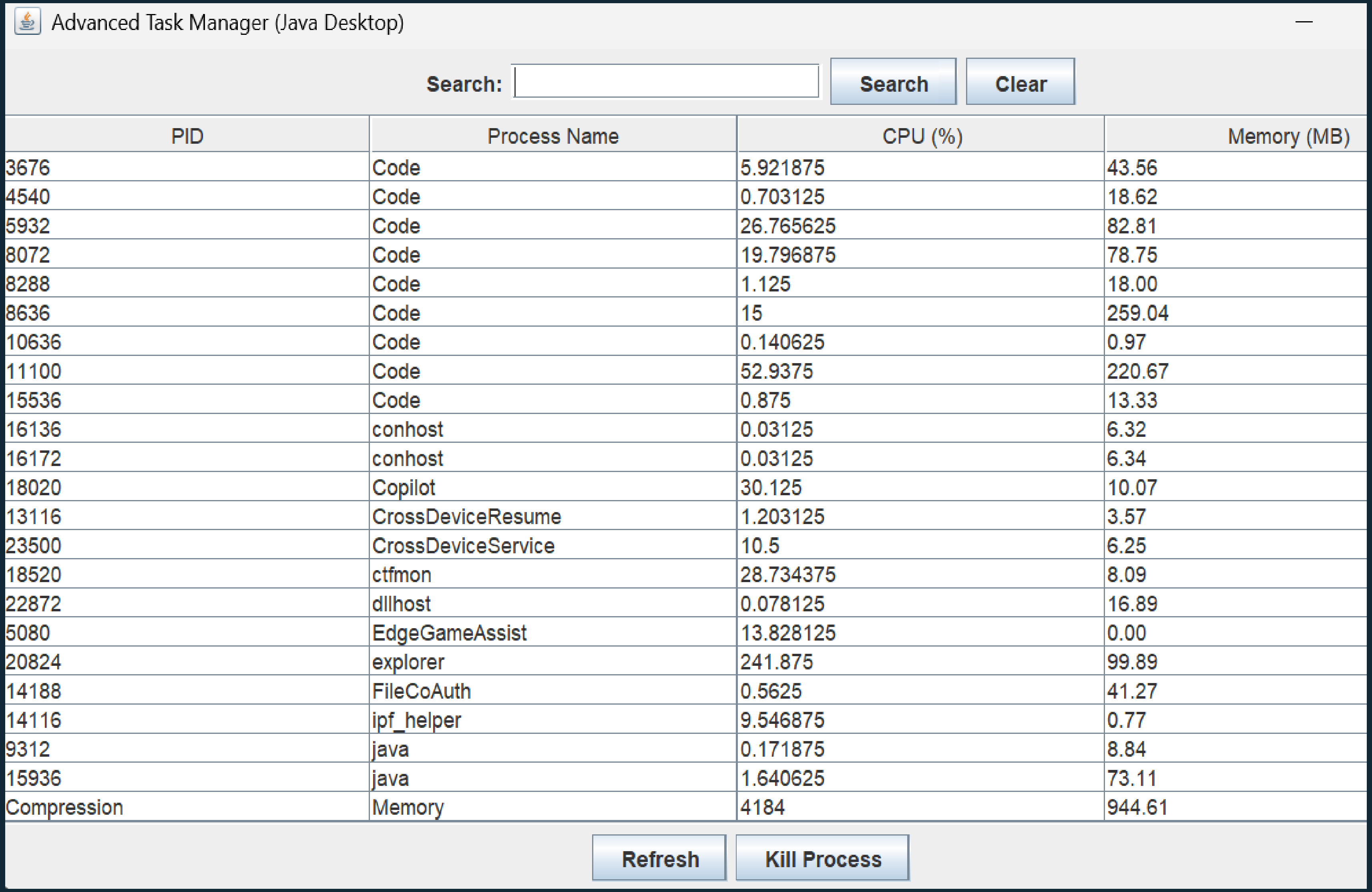
UI Updates

SwingUtilities ensures thread-safe GUI updates on the Event Dispatch Thread for smooth rendering.

Core Component Structure

Main Class Components

- **DefaultTableModel:** Manages table data with dynamic row addition and removal
- **JTable:** Displays process information in columns (PID, Name, CPU, Memory)
- **JTextField:** Captures search input for process filtering
- **JButton:** Triggers actions for refresh, kill, search, and clear operations
- **JScrollPane:** Enables scrolling through large process lists



PID	Process Name	CPU (%)	Memory (MB)
3676	Code	5.921875	43.56
4540	Code	0.703125	18.62
5932	Code	26.765625	82.81
8072	Code	19.796875	78.75
8288	Code	1.125	18.00
8636	Code	15	259.04
10636	Code	0.140625	0.97
11100	Code	52.9375	220.67
15536	Code	0.875	13.33
16136	conhost	0.03125	6.32
16172	conhost	0.03125	6.34
18020	Copilot	30.125	10.07
13116	CrossDeviceResume	1.203125	3.57
23500	CrossDeviceService	10.5	6.25
18520	ctfmon	28.734375	8.09
22872	dllhost	0.078125	16.89
5080	EdgeGameAssist	13.828125	0.00
20824	explorer	241.875	99.89
14188	FileCoAuth	0.5625	41.27
14116	ipf_helper	9.546875	0.77
9312	java	0.171875	8.84
15936	java	1.640625	73.11
Compression	Memory	4184	944.61

Process Loading Implementation

The loadProcesses() method is the heart of the application, executing PowerShell commands to retrieve real-time process data from the Windows operating system.

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

Execute Command

Runtime executes PowerShell Get-Process cmdlet

Parse Output

BufferedReader reads and splits process data

Populate Table

Data rows added to DefaultTableModel

Memory values are converted from bytes to megabytes for readability using: memBytes / (1024.0 * 1024.0)

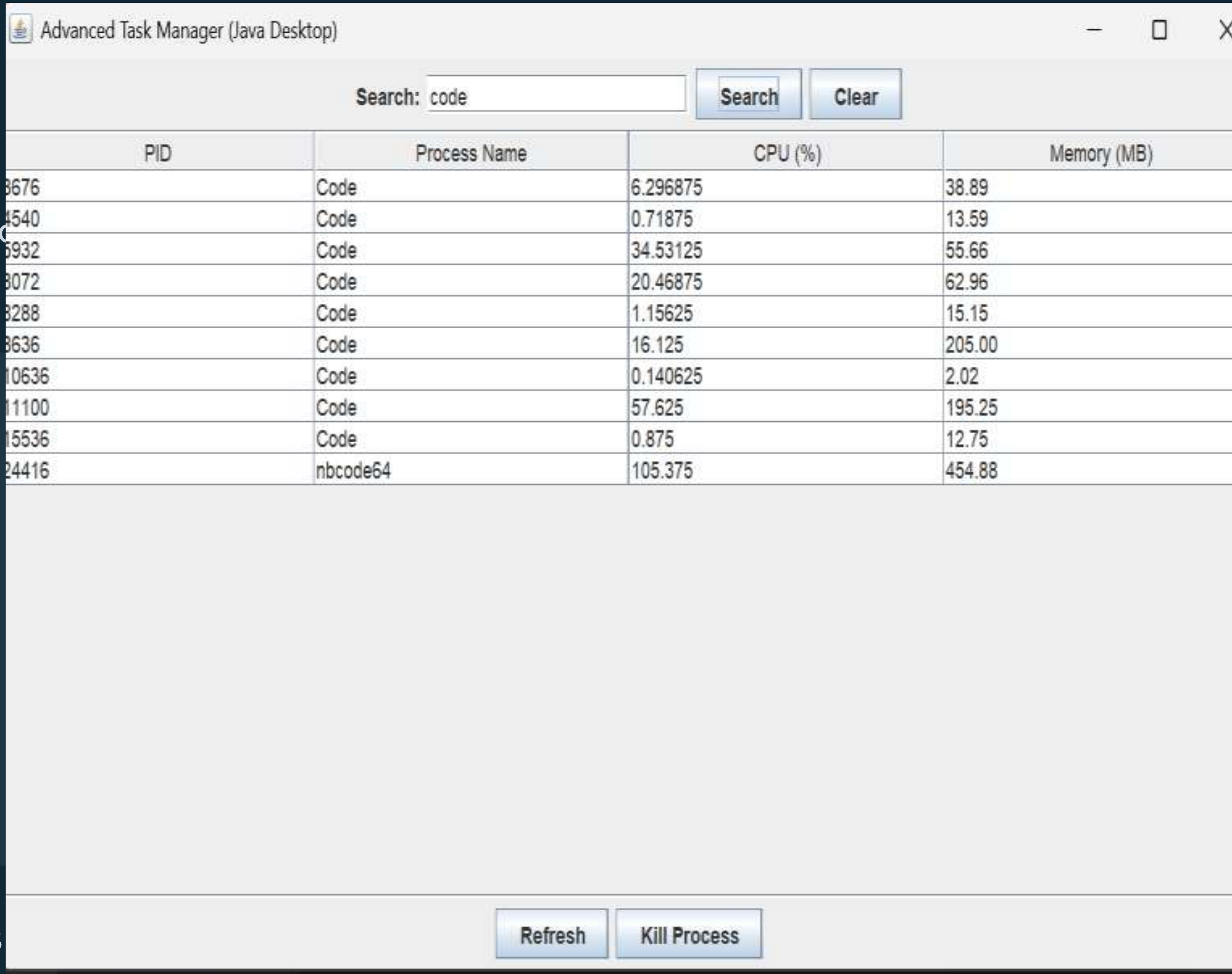
Search and Filter Functionality

Dynamic Process Filtering

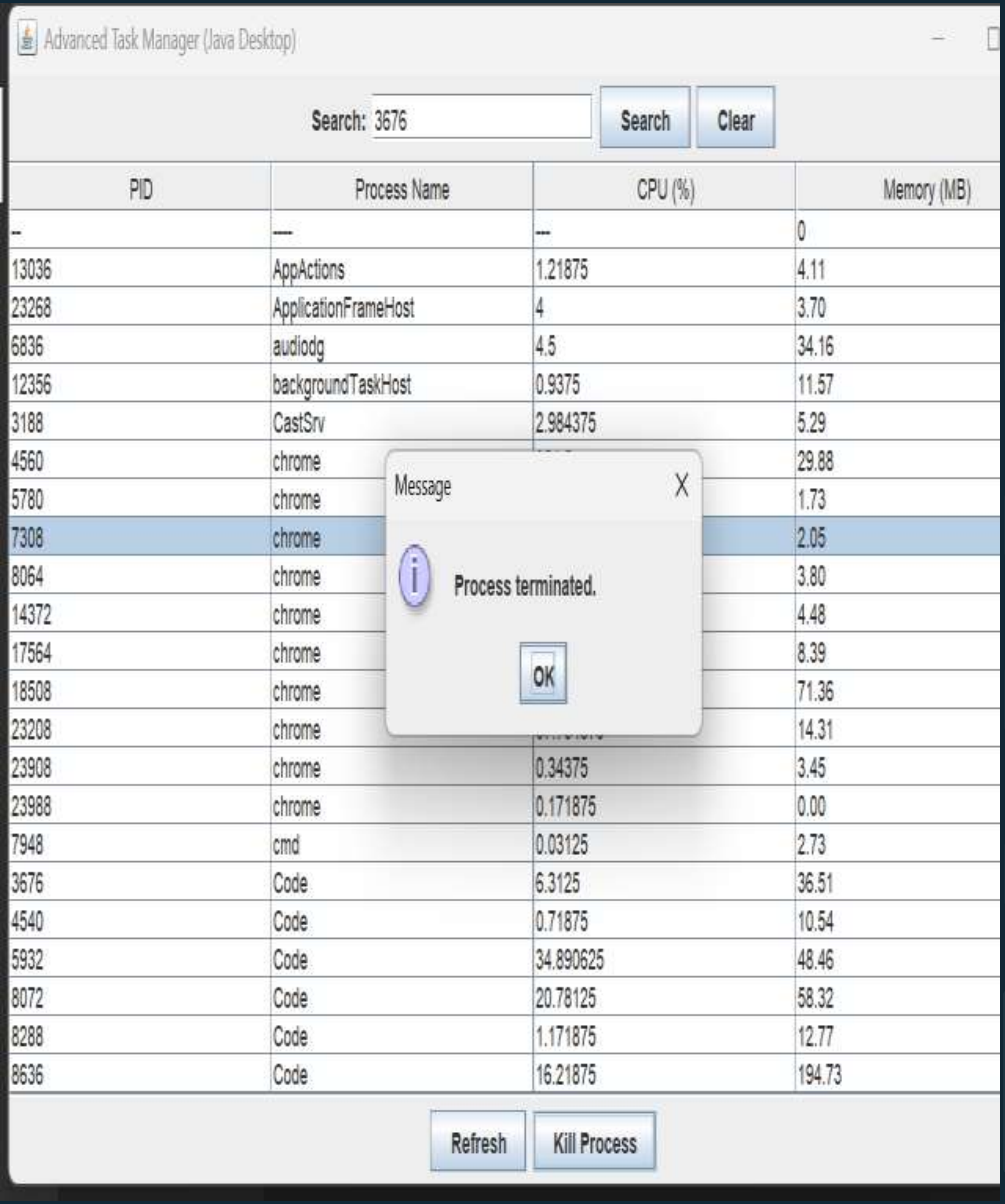
The searchProcess() method enables users to locate specific processes by name or PID. It reloads the complete process list and then iterates backwar through the table, removing rows that don't match the search criteria.

```
for (int i = tableModel.getRowCount() - 1; i >= 0; i--)
{  String name = tableModel.getValueAt(i, 1)
.toString().toLowerCase();  String pid =
tableModel.getValueAt(i, 0).toString();  if
(!(name.contains(keyword) || pid.contains(keyword))) {
tableModel.removeRow(i);  }}
```

The backwards iteration prevents index shifting issues when removing rows during traversal.



PID	Process Name	CPU (%)	Memory (MB)
3676	Code	6.296875	38.89
4540	Code	0.71875	13.59
5932	Code	34.53125	55.66
3072	Code	20.46875	62.96
3288	Code	1.15625	15.15
3636	Code	16.125	205.00
10636	Code	0.140625	2.02
11100	Code	57.625	195.25
15536	Code	0.875	12.75
24416	nbcode64	105.375	454.88



Advanced Task Manager (Java Desktop)

Search: 3676

PID	Process Name	CPU (%)	Memory (MB)
--	----	---	0
13036	AppActions	1.21875	4.11
23268	ApplicationFrameHost	4	3.70
6836	audiodg	4.5	34.16
12356	backgroundTaskHost	0.9375	11.57
3188	CastSrv	2.984375	5.29
4560	chrome	[illegible]	29.88
5780	chrome	[illegible]	1.73
7308	chrome	[illegible]	2.05
8064	chrome	[illegible]	3.80
14372	chrome	[illegible]	4.48
17564	chrome	[illegible]	8.39
18508	chrome	[illegible]	71.36
23208	chrome	[illegible]	14.31
23908	chrome	0.34375	3.45
23988	chrome	0.171875	0.00
7948	cmd	0.03125	2.73
3676	Code	6.3125	36.51
4540	Code	0.71875	10.54
5932	Code	34.890625	48.46
8072	Code	20.78125	58.32
8288	Code	1.171875	12.77
8636	Code	16.21875	194.73

Message

Process terminated.

Process Termination Feature

The killSelectedProcess() method provides users with the capability to forcefully terminate any selected process, functioning similarly to Windows Task Manager's "End Task" feature.

1 **Selection Validation**

Checks if a process row is selected in the table

2 **PID Extraction**

Retrieves the Process ID from the selected row

3 **Command Execution**

Executes taskkill /PID {pid} /F command

4 **Table Refresh**

Reloads process list to reflect changes

```
Runtime.getRuntime().exec("taskkill /PID " + pid + "
/F");JOptionPane.showMessageDialog(this, "Process terminated.");loadProcesses();
```

Threading and GUI Best Practises

Event Dispatch Thread

All GUI operations are executed on the EDT using SwingUtilities.invokeLater() to prevent threading issues and ensure smooth UI rendering. This is crucial for maintaining responsiveness during system command execution.

```
SwingUtilities.invokeLater(
() ->   new
ProcessManager().setVisible
(true));
```

Action Listeners

Button actions are attached using lambda expressions for clean, maintainable code. Each button triggers specific methods: refreshBtn loads processes, killBtn terminates selected process, searchBtn filters results.

```
refreshBtn.addActionListene
r(e ->
loadProcesses());killBtn.ad
dActionListener(e ->
killSelectedProcess());
```

Error Handling

IOException is caught during process execution and file operations. User-friendly error messages are displayed via JOptionPane to ensure graceful failure handling and improved user experience.

Learning Outcomes and Applications

System Command Execution

Mastering Runtime.getRuntime().exec() for executing operating system commands and processing their output streams in Java applications.

GUI Development with Swing

Building responsive desktop interfaces using JFrame, JTable, JPanel, and layout managers for professional application design.

Multithreading and EDT

Understanding Event Dispatch Thread principles and implementing thread-safe GUI updates using SwingUtilities for optimal performance.

This project demonstrates practical integration of Java GUI programming with system-level operations, providing a foundation for building robust desktop utilities and system monitoring tools on Windows platforms.

THANK YOU