Nathan Hull

CIS 452-20
Program 3: Simulated Paging System
Deliverable

Design Document

Summary

This program is designed to simulate a UNIX paging system for memory management. As processes are run, they must be brought into memory. This program was written using the MVC design paradigm to ensure a high degree of decoupling for ease of testing, and freedom of design.

Model

In this system, processes are described by a Process Control Block (PCB) with overall information such as the PID, page table, and total data length. The page table is made up of Frames, which contain their PID, type (data or text), page number, and size. Some of these parameters weren't used in this implementation, but were included for extensibility as this project is interesting with room to grow.

Algorithm

Frames in physical memory are identified by their index, as the physical memory is represented by an index of Frames objects. Thus, the Free Frame List is a queue (Linked List) of indices that are free, in the order they were freed. The first thing the program will do is check for empty frames in physical memory from 0 to the max, and add pages to those frames as it finds them. If there are still pages left over that need to be added, the program begins polling from the queue, removing indices from the head and replacing those frames with the new loading pages.

When creating a new process, the Frame objects (pages) are created sequentially by decrementing the amount of bytes in the "frame size" variable from the amount of data or text bytes, creating a Frame of that size and that type, then adding it to the PCB's page table. The last one will likely not be the full size a frame allows.

Controller

The Controller (a Singleton object) contains information like (dynamic) stats such as physical memory, frame size, and number of pages. It also contains a queue to store the free frame list via index in the physical memory block, which is represented by an array. The queue allows frames to be popped off in order, and by implementing free frame recovery by allowing new references to take frames off the queue, this will ensure that only the frames that have not been used in the longest time will be removed to make room for others. Finally, the Controller holds a list of processes currently in memory via an array list of their PCBs.

Directions

Upon startup, the user is given a safe screen as shown in Fig 0 – the Next button is not enabled until they've chosen an input file with data, and they can set different physical memory amounts or frame sizes using the textboxes in the top right (some issues with exact implementation, to be shown in demonstration). Once they've chosen a file, they may begin clicking the Next button to begin reading the file line by line. The process that was added or halted shows up with its page table in the left-hand table labeled Process Memory, and the changes to physical memory show up in the table to the right. When a process is halted, its page table is cleared and all the frames in physical memory are marked as free frames with an asterisk as shown in Fig 3. Finally, when another process comes in needing room for its pages, free frames in physical memory are replaced in the order they were freed. The end of the file is handled gracefully, stopping progress with a dialogue box.

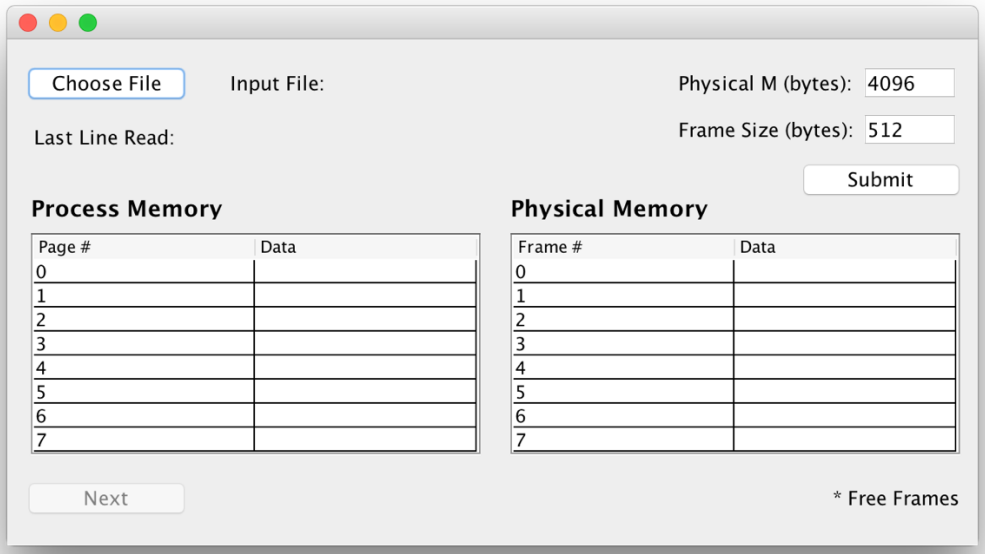

Fig 0: Initial startup screen

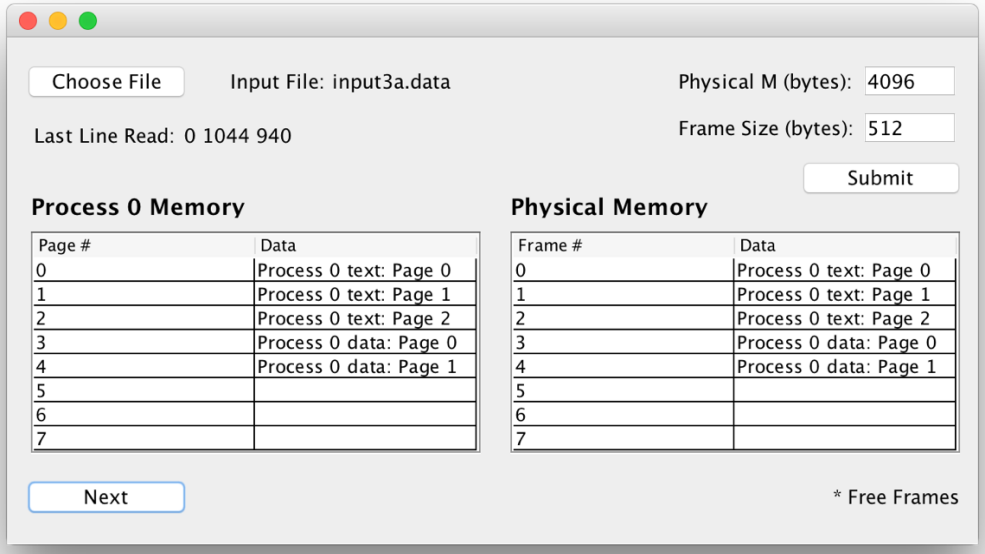

Fig 1: First input

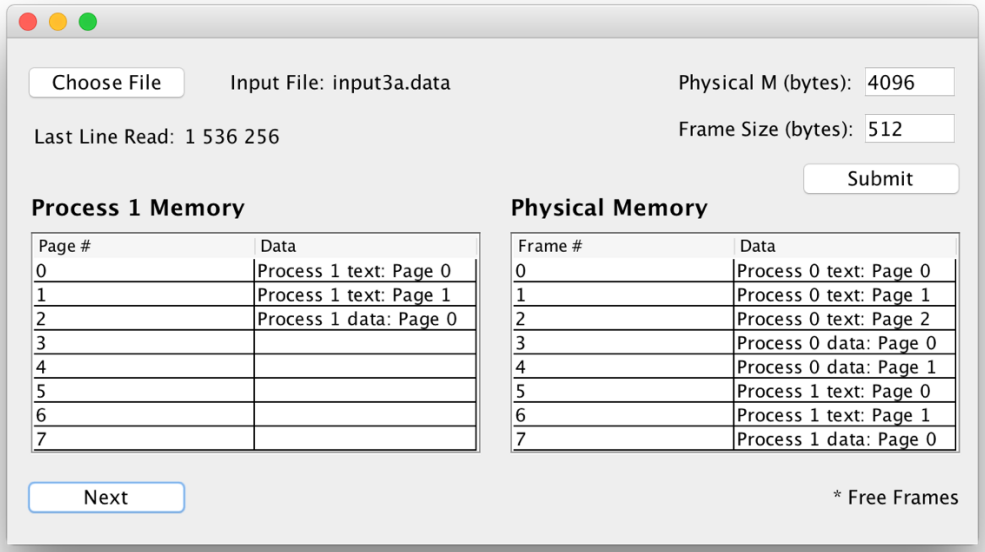

Fig 2: Second input

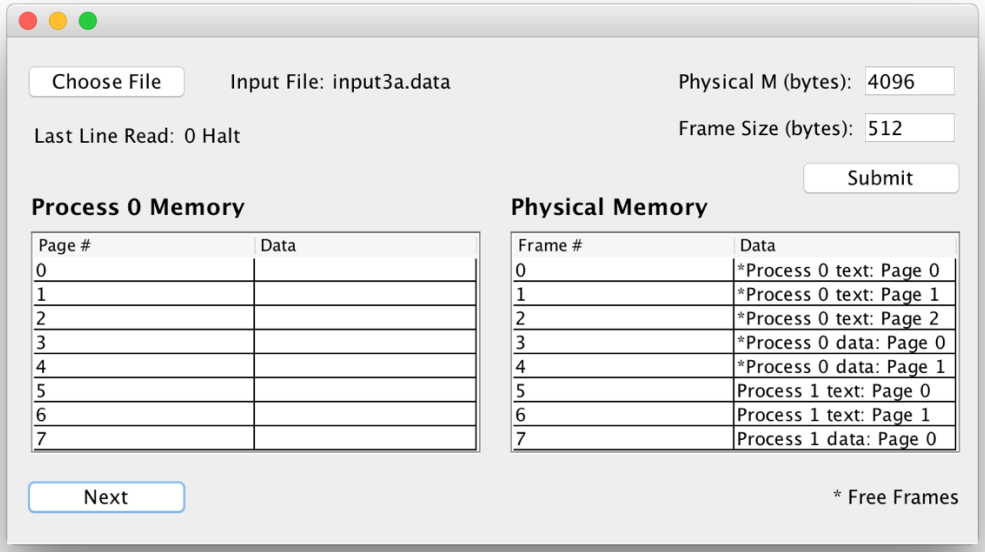

Fig 3: Free Frame Markers

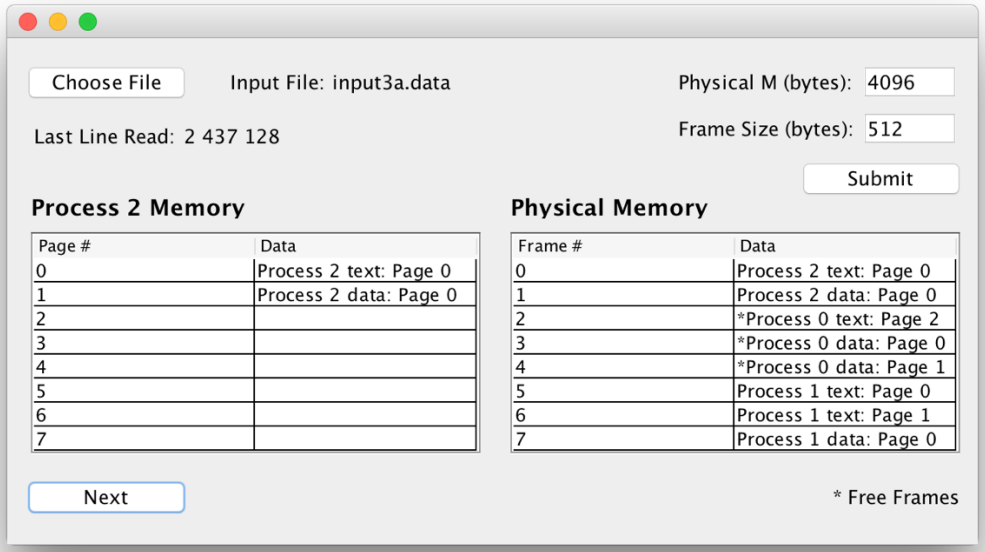

Fig 4: Overwriting of Free Frames

Frame.java

```
package project3;

public class Frame {
	private String pid;
	private String type;
	private int page;
	private int frame;
	private int size;

	/**
	* @return the pid
	*/
	public String getPid() {
		return pid;
	}
	/**
	* @param pid the pid to set
	*/
	public void setPid(String pid) {
		this.pid = pid;
	}
	/**
	* @return the type
	*/
	public String getType() {
		return type;
	}
	/**
	* @param type the type to set
	*/
	public void setType(String type) {
		this.type = type;
	}
	/**
	* @return the page
	*/
	public int getPage() {
		return page;
	}
	/**
	* @param page the page to set
	*/
	public void setPage(int page) {
		this.page = page;
	}
	/**
	* @return the frame
	*/
	public int getFrame() {
		return frame;
	}
	/**
	* @param frame the frame to set
```

```
	*/
	public void setFrame(int frame) {
		this.frame = frame;
	}
	/**
	* @return the size
	*/
	public int getSize() {
		return size;
	}
	/**
	* @param size the size to set
	*/
	public void setSize(int size) {
		this.size = size;
	}
	/**
	* @return string format of Frame
	*/
	public String toString() {
		StringBuilder result = new StringBuilder();
		result.append("Process ");
		result.append(pid);
		result.append(" ");
		result.append(type);
		result.append(": Page ");
		result.append(page);
		result.append(", Frame ");
		result.append(frame);

		return result.toString();
	}
}
```

ProcessControlBlock.java

```
package project3;

import java.util.ArrayList;

public class ProcessControlBlock {
	private String pid;
	private ArrayList<Frame> pageTable = new ArrayList<>();
	private int length;

	public ProcessControlBlock() {}

	/**
	* @return the pid
	*/
	public String getPid() {
		return pid;
	}
	/**
	* @param pid the pid to set
	*/
	public void setPid(String pid) {
		this.pid = pid;
	}
	/**
	* @return the pageTable
	*/
	public ArrayList<Frame> getPageTable() {
		return pageTable;
	}
	/**
	* @param pageTable the pageTable to set
	*/
	public void setPageTable(ArrayList<Frame> pageTable) {
		this.pageTable = pageTable;
	}
	/**
	* @return pageTable size
	*/
	public int getPageTableSize() {
		return pageTable.size();
	}
	/**
	* @return the length
	*/
	public int getLength() {
		return length;
	}
	/**
	* @param length the length to set
	*/
	public void setLength(int length) {
		this.length = length;
```

```
	}

	/**
	* Helper method to easily add frames to page table
	*
	* @param frame the frame to add to the page table
	*/
	public void appendFrame(Frame frame) {
		pageTable.add(frame);
	}
}
```

Controller.java

```
package project3;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;

public final class Controller {

	// Instance of Singleton Controller
	private static final Controller INSTANCE = new Controller();

	// Constant variables for sizes
	public int PHYSICAL_MEMORY_BYTES = 4096;
	public int FRAME_SIZE_BYTES = 512;
	public int NUMBER_OF_PAGES = PHYSICAL_MEMORY_BYTES/FRAME_SIZE_BYTES;

	// Queue for free frame list, stores index (location)
	// of free frames in physical memory in order
	private static Queue<Integer> freeFrameList = new LinkedList<>();

	// Array for physical memory table
	private static Frame physicalMemory[] = new Frame[INSTANCE.NUMBER_OF_PAGES];

	// File reader for file currently being read
	private static BufferedReader br;

	// Main form, aka view
	private static FrmMain frmMain;

	// List of processes currently in memory
	private static ArrayList<ProcessControlBlock> processes = new ArrayList<>();


	/*
	 * Empty as initialized final static instance for Singleton
	 */
	public Controller() {}

	/*
	 * Return final static instance of the controller
	 */
	public static Controller getInstance() {
		return INSTANCE;
```

```
	}

	/*
	 * Create and display frmMain
	 */
    private static void createAndShowGUI() {
        // Create and display window.
                    SwingUtilities.invokeLater(new Runnable() {
                            @Override
                            public void run() {
                                    try {
                                            frmMain = new FrmMain();
                                            frmMain.setVisible(true);
                                    } catch (Exception e) {
                                            e.printStackTrace();
                                    }
                            }
                    });
    }

    /*
     * The controller is also the driver
     */
    public static void main(String[] args) {
        // Create and display GUI
        createAndShowGUI();
    }

    /*
     * Called when user selects an input file
     */
    public void loadFile(File f) {
        if (f.isFile() && f.canRead()) {
                try {
                        // Close file if one is already open
                        if (br != null && br.ready())
                                br.close();
                                FileReader fr = new FileReader(f);
                                br = new BufferedReader(fr);
                        } catch (FileNotFoundException e) {
                                e.printStackTrace();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
        }
    }

    /*
     * Called when Next button is clicked, advance one line
     * in file and add it to menu
     */
    public void increment() {
        String line = "";
        try {
                        line = br.readLine();
```

```
                        // Catch EOF - EARLY RETURN
                        if (line == null) {
                                JOptionPane.showMessageDialog(frmMain, "End of file reached.");
                                return;
                        }

                        System.out.println("LINE: " + line);
                        frmMain.setLastLineReadText(line);
                        String elements[] = line.split(" ");
                        ProcessControlBlock pcb = new ProcessControlBlock();

                        if (elements[1].toLowerCase().equals("halt")) {
                                // Received halt command, add all frames in
                                // physical memory matching this pid to free list
                                for (int x = 0; x < physicalMemory.length; x++) {
                                        if (physicalMemory[x] != null &&
physicalMemory[x].getPid().equals(elements[0])) {
                                                if (!freeFrameList.contains(x))
                                                        freeFrameList.offer(x);
                                        }
                                }

                                // PCB is left mostly null to tell view to clear
                                pcb.setPid(elements[0]);

                        } else {
                                // Adding new process data/text to memory,
                                // first create its PCB
                                String pid = elements[0];
                                int textBytes = Integer.parseInt(elements[1]);
                                int dataBytes = Integer.parseInt(elements[2]);

                                pcb = new ProcessControlBlock();
                                pcb.setPid(elements[0]);
                                pcb.setLength(Integer.parseInt(elements[1]) +
Integer.parseInt(elements[2]));

                                // Then add frames to PCB page tables
                                int pageNumber = 0;
                                while (textBytes != 0) {
                                        Frame frame = new Frame();
                                        frame.setPid(pid);
                                        frame.setType("text");
                                        frame.setPage(pageNumber);

                                        // Decrement amount of bytes left by
                                        // size of frame, or remainder if smaller
                                        System.out.println("Text size: " + textBytes);
                                        frame.setSize(Math.min(FRAME_SIZE_BYTES, textBytes));
                                        textBytes -= Math.min(FRAME_SIZE_BYTES, textBytes);
                                        pageNumber++;

                                        // Add frame to PCB page table
                                        pcb.appendFrame(frame);
```

```
                                }

                                pageNumber = 0;
                                while (dataBytes != 0) {
                                        Frame frame = new Frame();
                                        frame.setPid(pid);
                                        frame.setType("data");
                                        frame.setPage(pageNumber);

                                        frame.setSize(Math.min(FRAME_SIZE_BYTES, dataBytes));
                                        dataBytes -= Math.min(FRAME_SIZE_BYTES, dataBytes);
                                        pageNumber++;

                                        pcb.appendFrame(frame);
                                }

                                processes.add(pcb);

                                // Finally, add process page table to memory

                                // Ctr to keep track of number of pages added
                                // to physical memory
                                int pagesAdded = 0;

                                // First try filling empty spots
                                for (int x = 0; x < physicalMemory.length; x++) {
                                        if (physicalMemory[x] == null) {
                                                physicalMemory[x] =
pcb.getPageTable().get(pagesAdded);
                                                pcb.getPageTable().get(pagesAdded).setFrame(x);
                                                pagesAdded++;
                                                if (pagesAdded >= pcb.getPageTableSize())
                                                        break;
                                        }
                                }

                                // If not all pages have been added...
                                while (pagesAdded < pcb.getPageTableSize()) {
                                        Integer freeFrame =    freeFrameList.poll();
                                        if (freeFrame == null) {
                                                JOptionPane.showMessageDialog(frmMain, "MEMORY
OVERFLOW");
                                                frmMain.setNextEnabled(false);
                                                frmMain.updateTables(physicalMemory, freeFrameList,
pcb);
                                                return;
                                        }
                                        physicalMemory[freeFrame] =
pcb.getPageTable().get(pagesAdded);
                                        pcb.getPageTable().get(pagesAdded).setFrame(freeFrame);
                                        pagesAdded++;
                                }
                        }

                        // Update view
```

```
                frmMain.updateTables(physicalMemory, freeFrameList, pcb);
            } catch (IOException e) {
                e.printStackTrace();
            }
    }

    public void changeDimensions(int pm, int fs) {
        System.out.println("CHANGING DIMENSIONS");
        INSTANCE.PHYSICAL_MEMORY_BYTES = pm;
        INSTANCE.FRAME_SIZE_BYTES = fs;
        INSTANCE.NUMBER_OF_PAGES = PHYSICAL_MEMORY_BYTES/FRAME_SIZE_BYTES;

        System.out.println("New PM: " + PHYSICAL_MEMORY_BYTES);
        System.out.println("New fs: " + FRAME_SIZE_BYTES);
        System.out.println("New page: " + NUMBER_OF_PAGES);

        physicalMemory = new Frame[INSTANCE.NUMBER_OF_PAGES];
        freeFrameList = new LinkedList<>();

        frmMain.dispose();
        createAndShowGUI();
    }
}
```

FrmMain.java

```
package project3;

import java.awt.Color;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JFileChooser;

import java.awt.event.ActionListener;
import java.io.File;
import java.util.ArrayList;
import java.util.Queue;
import java.awt.event.ActionEvent;
import javax.swing.JTable;
import java.awt.Font;
import javax.swing.JFormattedTextField;

@SuppressWarnings("serial")
public class FrmMain extends JFrame {

	// Reference to Controller singleton
	final Controller controller = Controller.getInstance();

	public JPanel contentPane;
	public JLabel lblLastLineRead;
	public JLabel lblProcessMemory;

	String physicalData[][];
	String processData[][];


	/**
	* Launch the application.
	*/
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					FrmMain frame = new FrmMain();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}
```

```
/**
* Create the frame.
*/
public FrmMain() {
	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	setBounds(100, 100, 648, 359);
	contentPane = new JPanel();
	contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
	setContentPane(contentPane);
	contentPane.setLayout(null);

	// For selecting input file
	final JFileChooser fc = new JFileChooser();

	JLabel lblInputFileTitle = new JLabel("Input File: ");
	lblInputFileTitle.setBounds(149, 21, 67, 16);
	contentPane.add(lblInputFileTitle);

	JLabel lblInputFile = new JLabel("");
	lblInputFile.setBounds(217, 21, 157, 16);
	contentPane.add(lblInputFile);

	JButton btnNext = new JButton("Next");
	btnNext.setEnabled(false);
	btnNext.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			controller.increment();
		}
	});
	btnNext.setBounds(8, 292, 117, 29);
	contentPane.add(btnNext);

	// User clicks button to choose input file
	JButton btnChangeFile = new JButton("Choose File");
	btnChangeFile.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			fc.setCurrentDirectory(new File("/Downloads"));
			int returnVal = fc.showOpenDialog(FrmMain.this);

		 if (returnVal == JFileChooser.APPROVE_OPTION) {
		     File file = fc.getSelectedFile();
		     lblInputFile.setText(file.getName());
		     controller.loadFile(file);
		     btnNext.setEnabled(true);
		 }
		}
	});
	btnChangeFile.setBounds(8, 16, 117, 29);
	contentPane.add(btnChangeFile);

	JLabel lblLastLineReadTitle = new JLabel("Last Line Read: ");
	lblLastLineReadTitle.setBounds(18, 57, 98, 16);
	contentPane.add(lblLastLineReadTitle);

	lblLastLineRead = new JLabel("");
```

```
lblLastLineRead.setBounds(118, 57, 288, 16);
contentPane.add(lblLastLineRead);


// ====== PROCESS PAGE TABLE REPRESENTATION ======

// Add table data statically because editing Model crashes
// Eclipse on Mac, some black magic with AWT/SWT deadlocking..?
// https://www.eclipse.org/forums/index.php/t/273408/
String processColumnNames[] = {"Page #", "Data"};

// Data initalized globally so it can be changed from controller
processData = new String[controller.NUMBER_OF_PAGES][2];
for (int x = 0; x < processData.length; x++) {
	processData[x][0] = "" + x;
	processData[x][1] = "";
}

JScrollPane spProcessMemory = new JScrollPane();
spProcessMemory.setBounds(16, 129, 300, 148);
JTable tblProcessMemory = new JTable(processData, processColumnNames);
tblProcessMemory.setGridColor(Color.BLACK);
spProcessMemory.setViewportView(tblProcessMemory);
contentPane.add(spProcessMemory);

lblProcessMemory = new JLabel("Process Memory");
lblProcessMemory.setFont(new Font("Lucida Grande", Font.BOLD, 15));
lblProcessMemory.setBounds(16, 104, 147, 16);
contentPane.add(lblProcessMemory);


// ====== PHYSICAL TABLE REPRESENTATION ======
String physicalColumnNames[] = {"Frame #", "Data"};

// Data initialized globally so it can be changed from controller
physicalData = new String[controller.NUMBER_OF_PAGES][2];
for (int x = 0; x < physicalData.length; x++) {
	physicalData[x][0] = "" + x;
	physicalData[x][1] = "";
	System.out.println("" + x);
}

JScrollPane spPhysicalMemory = new JScrollPane();
spPhysicalMemory.setBounds(336, 129, 300, 148);
JTable tblPhysicalMemory = new JTable(physicalData, physicalColumnNames);
tblPhysicalMemory.setGridColor(Color.BLACK);
spPhysicalMemory.setViewportView(tblPhysicalMemory);
contentPane.add(spPhysicalMemory);

JLabel lblPhysicalMemory = new JLabel("Physical Memory");
lblPhysicalMemory.setFont(new Font("Lucida Grande", Font.BOLD, 15));
lblPhysicalMemory.setBounds(336, 104, 147, 16);
contentPane.add(lblPhysicalMemory);

JLabel lblPS = new JLabel("* Free Frames");
```

```
		lblPS.setBounds(551, 297, 85, 16);
		contentPane.add(lblPS);

		JFormattedTextField txtPM = new JFormattedTextField();
		txtPM.setText("" + controller.PHYSICAL_MEMORY_BYTES);
		txtPM.setBounds(569, 16, 67, 26);
		contentPane.add(txtPM);

		JLabel lblPM = new JLabel("Physical M (bytes): ");
		lblPM.setBounds(448, 21, 122, 16);
		contentPane.add(lblPM);

		JLabel lblFrameSize = new JLabel("Frame Size (bytes): ");
		lblFrameSize.setBounds(448, 52, 122, 16);
		contentPane.add(lblFrameSize);

		JFormattedTextField txtFrameSize = new JFormattedTextField();
		txtFrameSize.setText("" + controller.FRAME_SIZE_BYTES);
		txtFrameSize.setBounds(569, 47, 67, 26);
		contentPane.add(txtFrameSize);

		JButton btnSubmit = new JButton("Submit");
		btnSubmit.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				controller.changeDimensions(Integer.parseInt(txtPM.getText()),
Integer.parseInt(txtFrameSize.getText()));
			}
		});
		btnSubmit.setBounds(525, 80, 117, 29);
		contentPane.add(btnSubmit);

		contentPane.repaint();
	}

	public void updateTables(Frame physicalMemory[], Queue<Integer> freeFrameList,
ProcessControlBlock pcb) {
		System.out.println("UPDATING");
		// Clear process table
		for (int x = 0; x < processData.length; x++) {
			processData[x][1] = "";
		}

		if (pcb.getPageTable() == null) {
			// No page table initialized, i.e. a process was halted
			lblProcessMemory.setText("Process " + pcb.getPid() + " Memory");
		} else {
			// Otherwise, load processor data in
			lblProcessMemory.setText("Process " + pcb.getPid() + " Memory");
			ArrayList<Frame> frames = pcb.getPageTable();
			for (int x = 0; x < pcb.getPageTableSize(); x++) {
				processData[x][1] = frames.get(x).toString();
			}
		}

		for (int x = 0; x < physicalData.length; x++) {
```

```
                        if (physicalMemory[x] == null) {
                                physicalData[x][1] = "";
                        } else {
                                if (freeFrameList.contains(x))
                                        physicalData[x][1] = '*' + physicalMemory[x].toString();
                                else
                                        physicalData[x][1] = physicalMemory[x].toString();
                        }
                }

                contentPane.repaint();
        }

        public void setLastLineReadText(String s) {
                lblLastLineRead.setText(s);
        }
}
```