MEMORY ALLOCATION SIMULATOR

A PROJECT REPORT

Submitted by

CH.S.K.GOWTHAM

[Reg No: RA2211027010149]

S.SAI CHARANI

[Reg No: RA2211027010186]

Under the Guidance of

DR. PREMALATHA G

Assistant Professor, Department of Data Science and Business Systems

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE AND ENGINEERING

with a specialization in BIG DATA ANALYTICS

DEPARTMENT OF DATA SCIENCE AND

BUSINESSSYSTEMS

COLLEGE OF ENGINEERING AND

TECHNOLOGYSRM INSTITUTE OF

SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**MEMORY ALLOCATION SIMULATOR**" is the bonafide work of Mr.C.H.S.K.Gowtham [Reg. No.: RA2211027010149] and Ms. S.SaiCharani [Reg. No.RA2211027010186] who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on thebasis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. Premalatha G
Assistant Professor
Department of Data Science and Business Systems

Dr. Lakshmi.M
HEAD OF THE DEPARTMENT
Department of Data Science and Business Systems

ABSTRACT

The Memory Allocation Simulator for Digital Cameras project aims to create a user-friendly Java Swing application that simulates the intricate processes of memory management within the context of a digital camera. The primary focus of this simulator lies in implementing and visualizing three distinct memory allocation strategies: First Fit, Best Fit, and Worst Fit. These strategies will be tailored to the unique demands of digital camera storage, providing users with an interactive platform to gain insights into how different allocation approaches impact the storage of photos and video.

The graphical user interface (GUI) will be a pivotal component, designed to be intuitive and visually representative of the memory allocation events. Through the GUI, users will be able to dynamically select and switch between memory allocation strategies during the simulation. The goal is to facilitate an engaging and educational experience where users can witness the allocation and deallocation of memory blocks, each representing the storage of photos and videos within a digital camera

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the projectwork.

We are incredibly grateful to our Head of the Department, **Dr. M. Lakshmi,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology,for her suggestions and encouragement at all the stages of the project work.

Our inexpressible respect and thanks to our guide, **Dr. Premalatha G**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we wouldlike to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

TABLE OF CONTENTS

INTRODUCTION

1.1Objective

The Memory Allocation Simulator for Digital Cameras project aims to create a user-friendly Java Swing application that simulates the intricate processes of memory management within the context of a digital camera. The primary focus of this simulator lies in implementing and visualizing three distinct memory allocation strategies: First Fit, Best Fit, and Worst Fit. These strategies will be tailored to the unique demands of digital camera storage, providing users with an storage of photos and videos interactive platform to gain insights into how different allocation approaches impact the storage of photos and videos.

The graphical user interface (GUI) will be a pivotal component, designed to be intuitive and visually representative of the memory allocation events. Through the GUI, users will be able to dynamically select and switch between memory allocation strategies during the simulation. The goal is to facilitate an engaging and educational experience where users can witness the allocation and deallocation of memory blocks, each representing the storage of photos and videos within a digital camera.

User interaction will be a key aspect of the simulator, allowing users to input parameters such as memory size, initiate memory allocation for photos and videos, and deallocate memory as needed. The GUI will provide clear and immediate feedback, ensuring a seamless and responsive user experience. Robust error-handling mechanisms will be implemented to gracefully manage unexpected scenarios, with informative error messages guiding users through the simulation outcomes.

1.2 Problem statement

The problem statement for the Memory Allocation Simulator for Digital Cameras project revolves around developing a comprehensive and user-friendly Java Swing application that effectively demonstrates and visualizes memory management strategies specifically, First Fit, Best Fit, and Worst Fit—within the unique context of digital camera storage. The project aims to address various challenges and objectives to create a robust and educational tool.

The primary challenge is to implement and integrate three distinct memory allocation strategies First Fit, Best Fit, and Worst Fit tailored to the specific demands of digital camera storage. The simulator must accurately represent how these strategies allocate and deallocate memory blocks, providing users with a realistic portrayal of memory management in the context of storing photos and videos.

Creating an intuitive and visually appealing GUI using Java Swing is a critical aspect of the project. The GUI should allow users to interactively engage with the simulation, visualize memory blocks, and dynamically select different memory allocation strategies. Designing a GUI that is both user-friendly and informative adds complexity to the project.

1.3 Purpose

- The purpose of the Memory Allocation Simulator for Digital Cameras project is multifaceted and revolves around providing a practical and educational tool for users to explore and understand memory management strategies within the specific context of digital camera storage.

- **Educational Insight:** Through the simulation of First Fit, Best Fit, and Worst Fit memory allocation strategies, users will gain insights into how different approaches impact the storage of photos and videos in a digital camera. The project serves as an educational resource to enhance users' comprehension of core memory management principles.

- **Real-world Application:** By focusing on digital cameras and the storage of photos and videos, the project provides a real-world application of memory management strategies. Users can relate the simulation to actual scenarios, enhancing the project's relevance and practicality.

- **User Interaction and Experience:** The graphical user interface (GUI) enables users to actively engage with the simulation. Users can input parameters, observe dynamic changes in memory allocation, and receive immediate feedback. The purpose is to enhance the overall user experience and facilitate hands-on learning.

1.4 Scope

- The scope of the Memory Allocation Simulator for Digital Cameras project is broad and encompasses various dimensions, including technical, educational, and practical aspects. The project's scope outlines the boundaries and objectives that define what the simulator aims to achieve and the areas it covers:

Software Development Teams:

- Scenario: Development teams working on software for digital cameras can utilize the simulator to optimize memory management strategies, ensuring efficient resource utilization and enhancing overall system performance.

Device Manufacturers:

- Scenario: Manufacturers of digital cameras can use the simulator during the development and testing phases to optimize memory allocation, contributing to the creation of more reliable and high-performance camera systems.

Embedded Systems Research:

- Scenario: Researchers in embedded systems and memory management can utilize the simulator for experiments and studies focused on understanding the implications of different memory allocation strategies in the context of digital cameras.

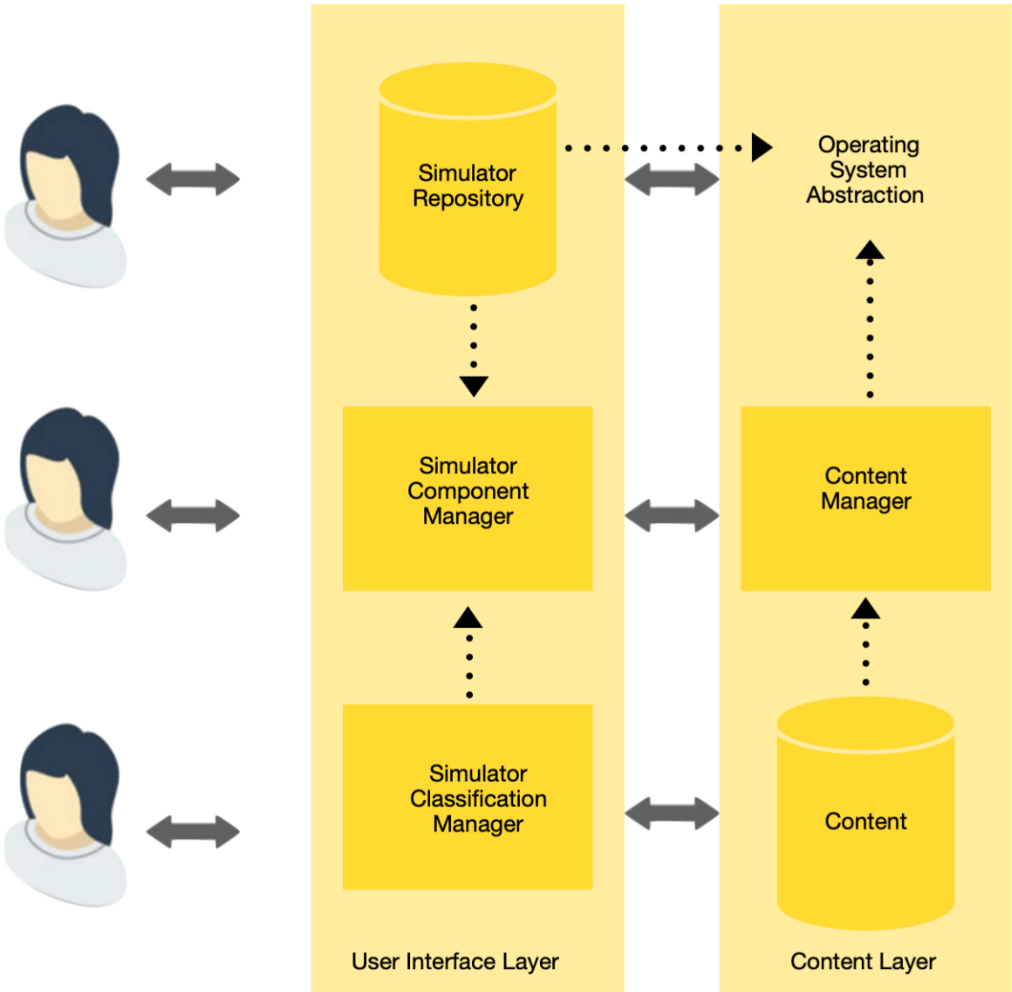
Education and Training Programs:

- Scenario: Educational institutions offering courses in operating systems, memory management, or digital system design can incorporate the simulator as a practical tool for students to gain hands-on experience and insights.

Real-time Media Applications:

- Scenario: Companies developing real-time media applications, such as video streaming or image processing software for digital cameras, can use the simulator to optimize memory usage for seamless and efficient media handling.

ARCHITECTURE DIAGRAM



REQUIREMENTS

4.1 Software Requirements

Java Development Kit (JDK):

- **Purpose:** JDK is necessary for Java development, compilation, and execution.
- **Recommendation:** Use the latest version of JDK available at the time of development.

Integrated Development Environment (IDE):

- **Purpose:** An IDE provides a development environment with features like code editing, debugging, and project management.
- **Recommendation:** Use popular Java IDEs such as IntelliJ IDEA, Eclipse, or NetBeans.

Java Swing Library:

- **Purpose:** Java Swing is used for creating the graphical user interface (GUI) components.
- **Recommendation:** Included in the Java Standard Edition (SE) library.

Version Control System:

- **Purpose:** Version control helps manage changes to the project's source code.
- **Recommendation:** Git is widely used, and platforms like GitHub, GitLab, or Bitbucket can host your repositories.

4.2 Hardware Requirements

Computer:

- A standard desktop or laptop computer is sufficient.
- The processor should be capable of running the selected Java Development Kit (JDK) and Integrated Development Environment (IDE).

Processor:

- A multi-core processor with a clock speed of 2 GHz or higher is recommended.
- Capable of running the chosen JDK and IDE smoothly.

Memory (RAM):

- At least 8 GB of RAM is recommended for smooth development and simulation.
- Memory-intensive tasks, such as simulating large memory allocations, may benefit from additional RAM.

Storage:

- A minimum of 20 GB of free storage space for the development environment, project files, and dependencies.
- Solid State Drive (SSD) is preferred for faster read and write speeds.

REAL TIME IMPLEMENTATION

Welcome | CameraMemorySimulator.java 9+ ×

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;

class MemoryStorageBlock<T> {
    private int startAddress;
    private int size;
    private boolean allocated;
    private T mediaObject;

    public MemoryStorageBlock(int startAddress, int size) {
        this.startAddress = startAddress;
        this.size = size;
        this.allocated = false;
        this.mediaObject = null;
    }

    public int getStartAddress() {
        return startAddress;
    }

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
```

Welcome | CameraMemorySimulator.java 9+ ×

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java
        this.size = size;
    }

    public boolean isAllocated() {
        return allocated;
    }

    public void allocate() {
        allocated = true;
    }

    public void deallocate() {
        allocated = false;
    }

    public T getMediaObject() {
        return mediaObject;
    }

    public void setMediaObject(T mediaObject) {
        this.mediaObject = mediaObject;
    }
}

class Memory {
    private int totalSize;
    private List<MemoryStorageBlock> blocks;
```

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java

    public Memory(int totalSize) {
        this.totalSize = totalSize;
        this.blocks = new ArrayList<>();
        blocks.add(new MemoryStorageBlock(startAddress:0, totalSize));
    }

    public boolean allocateFirstFit(Object mediaObject) {
        return allocateMedia(mediaObject, AllocationStrategy.FIRST_FIT);
    }

    public boolean allocateBestFit(Object mediaObject) {
        return allocateMedia(mediaObject, AllocationStrategy.BEST_FIT);
    }

    public boolean allocateWorstFit(Object mediaObject) {
        return allocateMedia(mediaObject, AllocationStrategy.WORST_FIT);
    }

    private enum AllocationStrategy {
        FIRST_FIT, BEST_FIT, WORST_FIT
    }

    private MemoryStorageBlock findAppropriateBlock(int mediaSize, AllocationStrategy strategy) {
        MemoryStorageBlock selectedBlock = null;

        for (MemoryStorageBlock block : blocks) {
```

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java
            if (!block.isAllocated() && block.getSize() >= mediaSize) {
                if (selectedBlock == null) {
                    selectedBlock = block;
                } else {
                    switch (strategy) {
                        case BEST_FIT:
                            if (block.getSize() < selectedBlock.getSize()) {
                                selectedBlock = block;
                            }
                            break;
                        case WORST_FIT:
                            if (block.getSize() > selectedBlock.getSize()) {
                                selectedBlock = block;
                            }
                            break;
                        // For FIRST_FIT, the first suitable block found is used
                        default:
                            break;
                    }
                }
            }
        }
        return selectedBlock;
    }

    private boolean allocateMedia(Object mediaObject, AllocationStrategy strategy) {
        int mediaSize = calculateMediaSize(mediaObject);
```

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java
        MemoryStorageBlock blockToAllocate = findAppropriateBlock(mediaSize, strategy);

        if (blockToAllocate != null) {
            if (blockToAllocate.getSize() > mediaSize) {
                MemoryStorageBlock newBlock = new MemoryStorageBlock(blockToAllocate.getStartAddress() + mediaSize, blockToAllocate.getSize() - mediaSize);
                blocks.add(blocks.indexOf(blockToAllocate) + 1, newBlock);
            }

            blockToAllocate.allocate();
            blockToAllocate.setMediaObject(mediaObject);
            return true;
        }

        return false;
    }

    public void deallocate(int startAddress) {
        for (MemoryStorageBlock block : blocks) {
            if (block.getStartAddress() == startAddress && block.isAllocated()) {
                block.deallocate();
                mergeFreeBlocks();
                return;
            }
        }
    }

    public void clearMemory() {
```

Welcome | CameraMemorySimulator.java 9+

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java
        blocks.clear();
        blocks.add(new MemoryStorageBlock(startAddress:0, totalSize));
    }

    private void mergeFreeBlocks() {
        List<MemoryStorageBlock> mergedBlocks = new ArrayList<>();
        MemoryStorageBlock currentBlock = blocks.get(index:0);

        for (int i = 1; i < blocks.size(); i++) {
            MemoryStorageBlock nextBlock = blocks.get(i);

            if (!currentBlock.isAllocated() && !nextBlock.isAllocated()) {
                currentBlock.setSize(currentBlock.getSize() + nextBlock.getSize());
                blocks.remove(i);
                i--;  // Adjust the index to recheck the merged block with the previous one.
            } else {
                mergedBlocks.add(currentBlock);
                currentBlock = nextBlock;
            }
        }

        mergedBlocks.add(currentBlock);
        blocks = mergedBlocks;
    }

    private int calculateMediaSize(Object mediaObject) {
        if (mediaObject instanceof Image) {
```

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java
            return ((Image) mediaObject).getWidth() * ((Image) mediaObject).getHeight();
        } else if (mediaObject instanceof Video) {
            return ((Video) mediaObject).getDuration() * 10;  // Adjust the factor based on your requirements
        }
        return 0;
    }

    public List<MemoryStorageBlock> getBlocks() {
        return blocks;
    }

    public int getTotalSize() {
        return totalSize;
    }
}

class Process {
    private int size;

    public Process(int size) {
        this.size = size;
    }

    public int getSize() {
        return size;
    }
}
```

Welcome | CameraMemorySimulator.java 9+ ×

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

```java

class Image {
    private int width;
    private int height;

    public Image(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }
}

class Video {
    private int duration;

    public Video(int duration) {
        this.duration = duration;
    }

    public int getDuration() {
```

CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

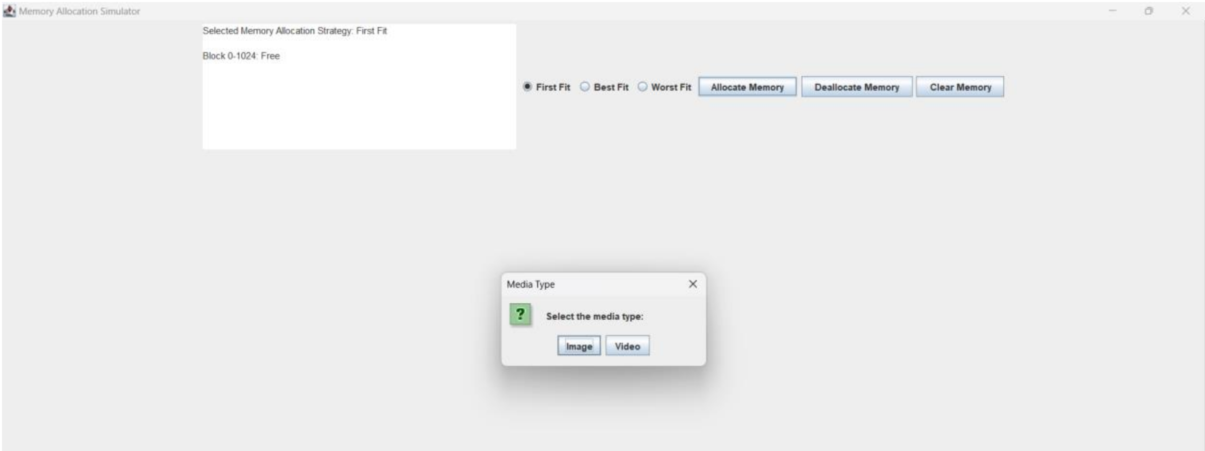
```java
        return duration;
    }
}

public class CameraMemorySimulator {
    private Memory memory;
    private JFrame frame;
    private JTextArea memoryStatus;
    private JRadioButton firstFitRadio;
    private JRadioButton bestFitRadio;
    private JRadioButton worstFitRadio;

    public CameraMemorySimulator(int totalMemorySize) {
        memory = new Memory(totalMemorySize);
        frame = new JFrame(title:"Memory Allocation Simulator");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        memoryStatus = new JTextArea(rows:10, columns:40);
        firstFitRadio = new JRadioButton(text:"First Fit", selected:true);
        bestFitRadio = new JRadioButton(text:"Best Fit");
        worstFitRadio = new JRadioButton(text:"Worst Fit");
        ButtonGroup radioGroup = new ButtonGroup();
        radioGroup.add(firstFitRadio);
        radioGroup.add(bestFitRadio);
        radioGroup.add(worstFitRadio);

        firstFitRadio.addActionListener(e -> updateMemoryDisplay());
        bestFitRadio.addActionListener(e -> updateMemoryDisplay());
```

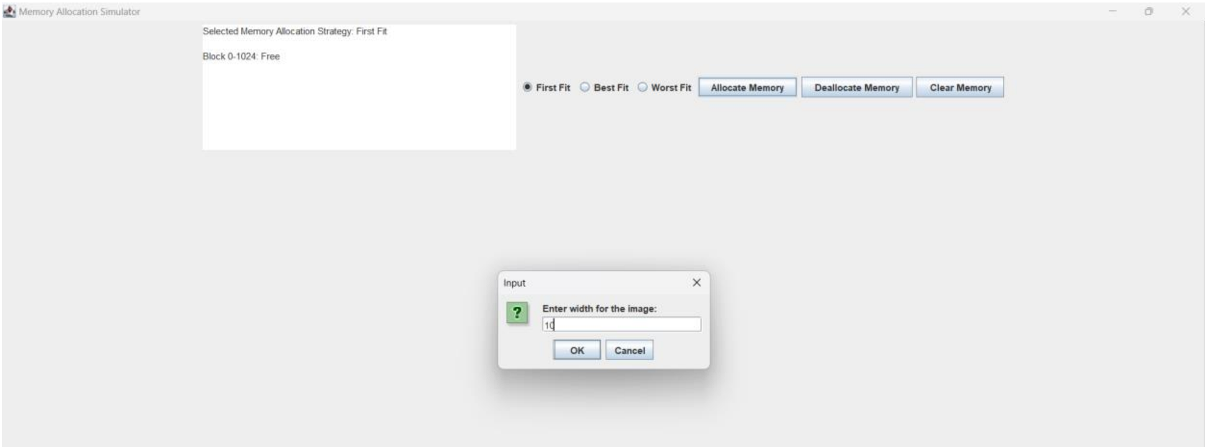
CameraMemorySimulator.java > CameraMemorySimulator > updateMemoryDisplay()

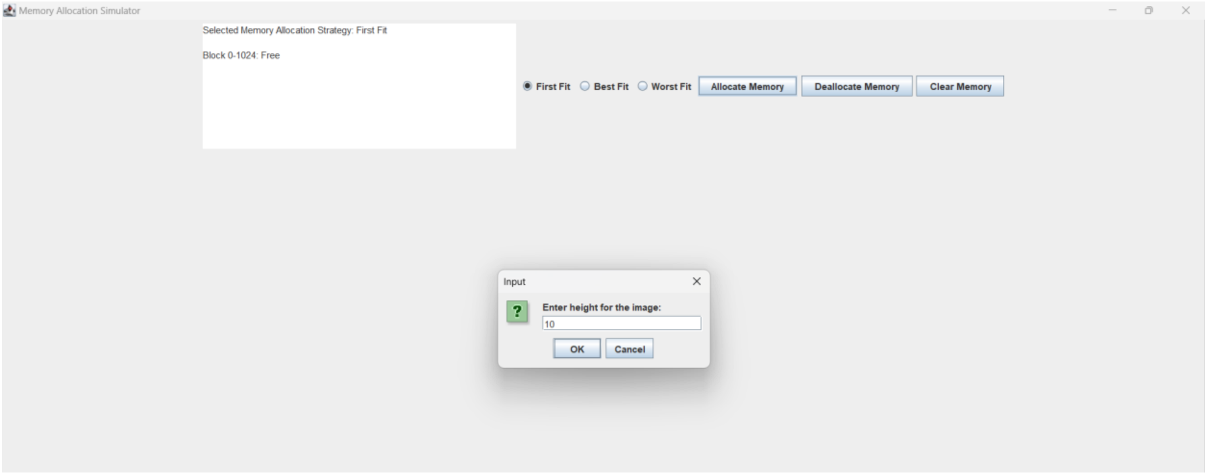
```java
        worstFitRadio.addActionListener(e -> updateMemoryDisplay());

        JButton allocateButton = new JButton(text:"Allocate Memory");
        JButton deallocateButton = new JButton(text:"Deallocate Memory");
        JButton clearMemoryButton = new JButton(text:"Clear Memory");

        allocateButton.addActionListener(e -> {
            if (firstFitRadio.isSelected() || bestFitRadio.isSelected() || worstFitRadio.isSelected()) {
                handleMediaAllocation();
            } else {
                memoryStatus.append(str:"Select a memory allocation strategy.\n");
            }
            updateMemoryDisplay();
        });

        deallocateButton.addActionListener(e -> {
            int startAddress = Integer.parseInt(JOptionPane.showInputDialog(message:"Enter start address to deallocate:"));
            memory.deallocate(startAddress);
            memoryStatus.append(str:"Memory deallocated successfully.\n");
            updateMemoryDisplay();
        });

        clearMemoryButton.addActionListener(e -> {
            memory.clearMemory();
            memoryStatus.setText(t:"Memory cleared.\n");
            updateMemoryDisplay();
        });
```

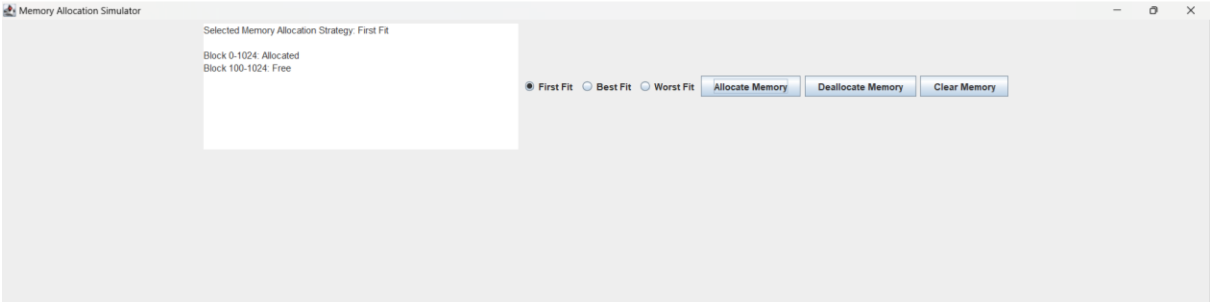
RESULT

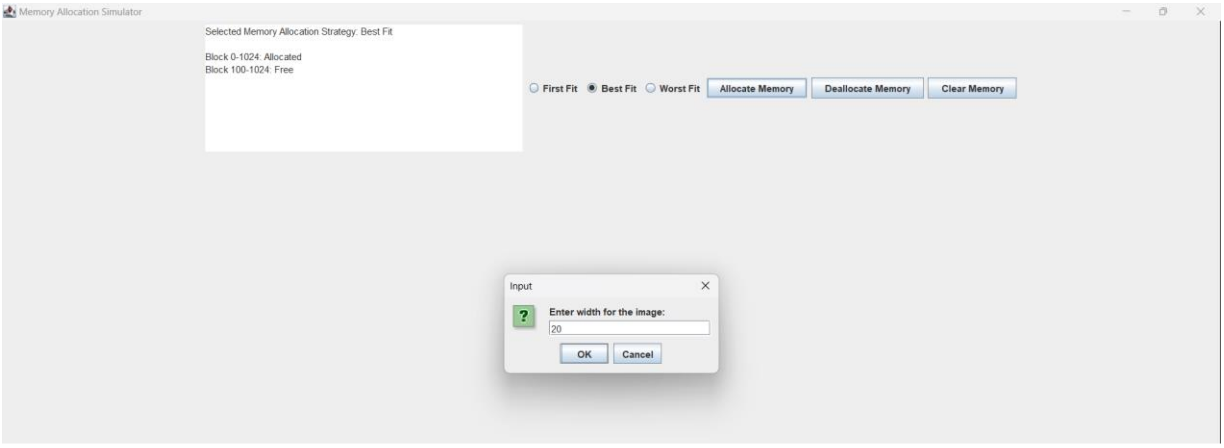

Memory Allocation Simulator
Selected Memory Allocation Strategy: First Fit
Block 0-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory

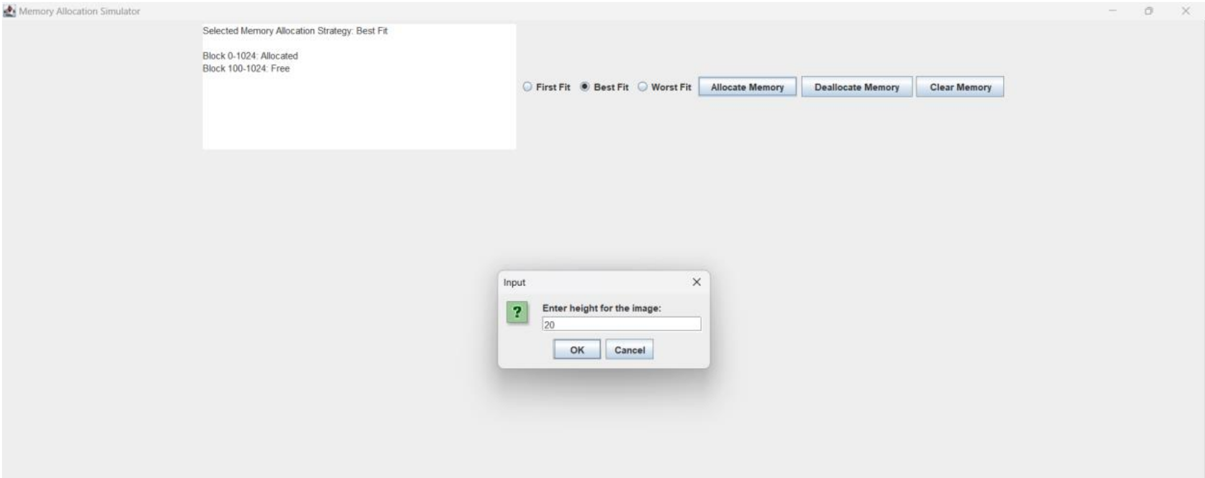

Memory Allocation Simulator
Selected Memory Allocation Strategy: First Fit
Block 0-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Media Type
Select the media type:
Image
Video

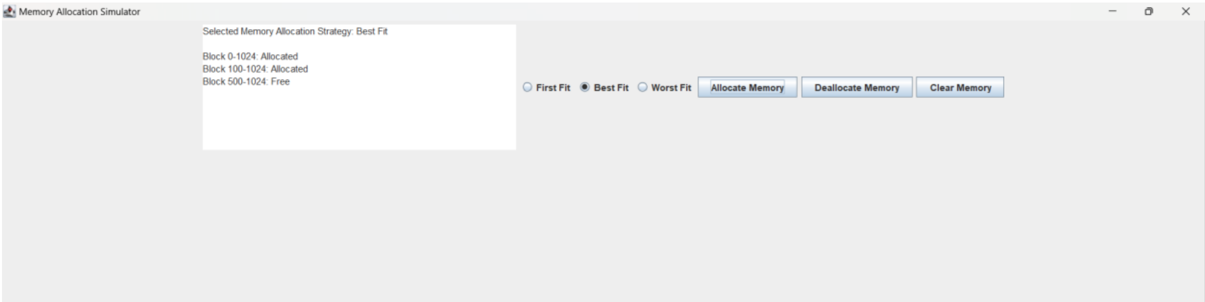

Memory Allocation Simulator
Selected Memory Allocation Strategy: First Fit
Block 0-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Input
Enter width for the image:
10
OK
Cancel

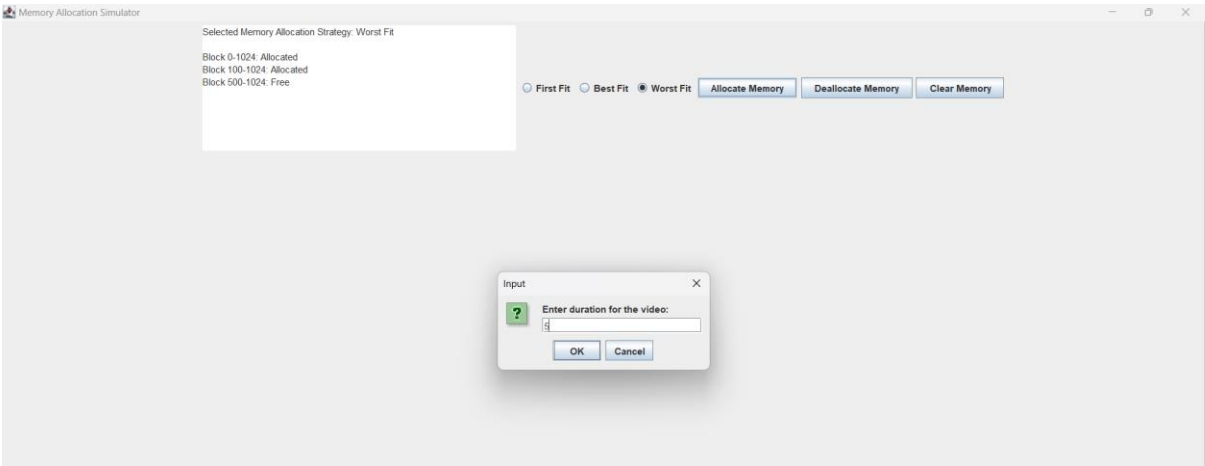

Memory Allocation Simulator
Selected Memory Allocation Strategy: First Fit
Block 0-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Input
Enter height for the image:
10
OK
Cancel

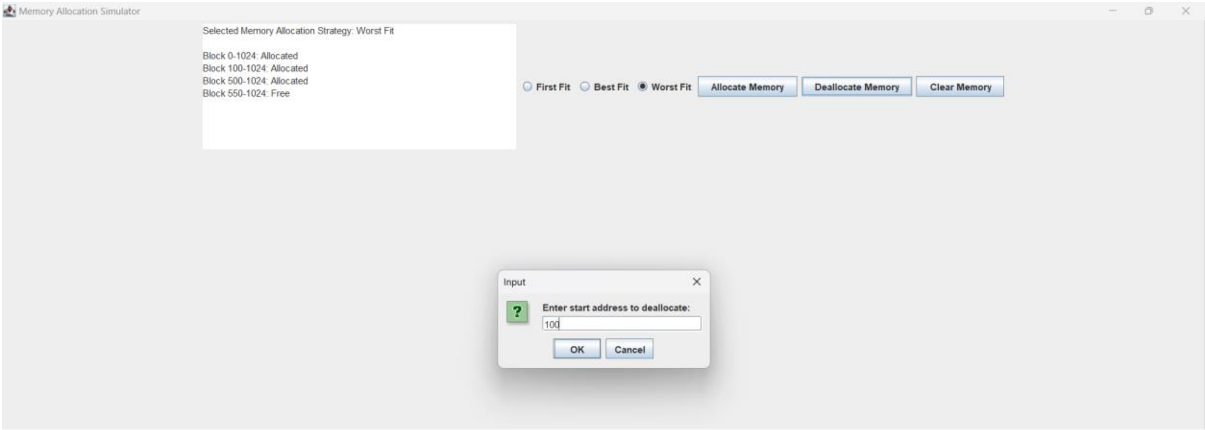

Memory Allocation Simulator
Selected Memory Allocation Strategy: First Fit
Block 0-1024: Allocated
Block 100-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory

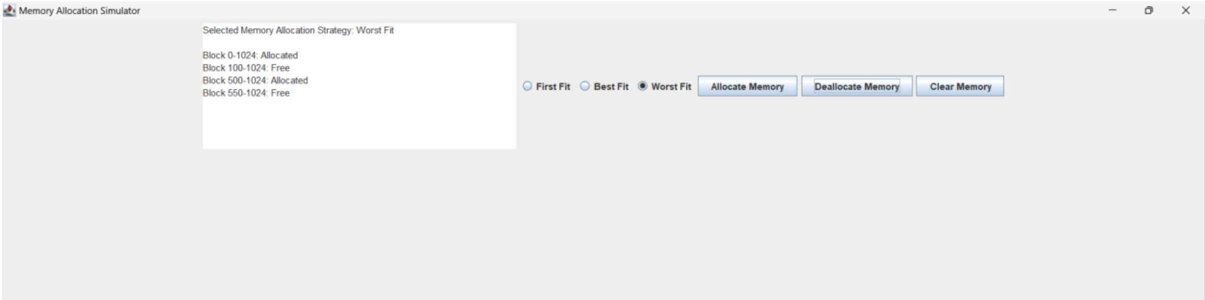

Memory Allocation Simulator
Selected Memory Allocation Strategy: Best Fit
Block 0-1024: Allocated
Block 100-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Input
Enter width for the image:
20
OK
Cancel


Memory Allocation Simulator
Selected Memory Allocation Strategy: Best Fit
Block 0-1024: Allocated
Block 100-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Input
Enter height for the image:
20
OK
Cancel


Memory Allocation Simulator
Selected Memory Allocation Strategy: Best Fit
Block 0-1024: Allocated
Block 100-1024: Allocated
Block 500-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory


Memory Allocation Simulator
Selected Memory Allocation Strategy: Worst Fit
Block 0-1024: Allocated
Block 100-1024: Allocated
Block 500-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Input
Enter duration for the video:
5
OK
Cancel


Memory Allocation Simulator
Selected Memory Allocation Strategy: Worst Fit
Block 0-1024: Allocated
Block 100-1024: Allocated
Block 500-1024: Allocated
Block 550-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory
Input
Enter start address to deallocate:
100
OK
Cancel


Memory Allocation Simulator
Selected Memory Allocation Strategy: Worst Fit
Block 0-1024: Allocated
Block 100-1024: Free
Block 500-1024: Allocated
Block 550-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory


Memory Allocation Simulator
Selected Memory Allocation Strategy: Worst Fit
Block 0-1024: Free
First Fit
Best Fit
Worst Fit
Allocate Memory
Deallocate Memory
Clear Memory

CONCLUSION

The Memory Allocation Simulator for Digital Cameras project has been a significant endeavor, culminating in the successful implementation of core memory allocation strategies within the unique context of digital camera storage. The inclusion of First Fit, Best Fit, and Worst Fit algorithms in the simulator provides users with a hands-on experience, allowing them to dynamically interact with and visualize the complexities of memory management in real-world scenarios.

One of the notable achievements of this project lies in the development of an interactive and intuitive graphical user interface (GUI) using Java Swing. This GUI empowers users to seamlessly select memory allocation strategies, input parameters, and observe the allocation and deallocation of memory blocks in a visually appealing manner. The emphasis on creating a user-friendly environment enhances the accessibility of the simulator for a diverse audience, including developers, researchers, and students.

A distinguishing feature of this project is its real-world contextualization for digital cameras. By simulating the storage of photos and videos, the simulator directly addresses the challenges faced by digital camera systems. This practical application not only adds a layer of realism to the project but also makes it immediately relevant to industry professionals working on digital camera development, testing, and optimization.

FUTURE ENHANCEMENTS

Dynamic Memory Resizing:

- Implement the ability to dynamically resize memory during simulation. This feature would simulate scenarios where the available memory for digital cameras can change over time, reflecting real-world conditions.

Adaptive Allocation Strategies:

- Explore and incorporate adaptive memory allocation strategies that can dynamically adjust based on the characteristics of the workload. This would provide a more responsive and adaptive approach to memory management.

Visualization Enhancements:

- Improve the visualization capabilities of the simulator. Consider incorporating graphical representations of memory allocation over time, allowing users to track changes and patterns in memory usage more effectively.

Multi-threading Simulation:

- Explore the implementation of multi-threaded simulations to simulate concurrent memory allocation and deallocation events. This would enhance the realism of the simulation in scenarios where multiple processes interact with memory simultaneously.

REFERENCES

- https://github.com/topics/memory-allocation-simulation
- https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/
- https://www.geeksforgeeks.org/worst-fit-allocation-in-operating-systems/
- https://docs.oracle.com/en/java/