DATA STRUCTURE COURSE DESIGN PROJECT REPORT

Course: Data Structure Course Design
Instructor: [Sajjad]
Semester: [Semester 1/2025]
Academic Year: [2025-2026]

PROJECT TITLE: [**ISMA File Compressor**]

TEAM MEMBERS

Student ID	Name	Contribution %	Signature
2024080156	Cafrida Soufyane		
2024080120	Ait lasiri Youssef		
2024003126	Barrar Mohamed		

SUBMISSION DATE: [27/10/25]
PROJECT DURATION: [20/10/25] - [End Date]

TABLE OF CONTENTS

1. ABSTRACT

[Provide a concise summary of your project in 150-200 words. Describe the main purpose, methods, data structures used, and key results.]

We developed ISMA File Compressor, a GUI-based application designed to intelligently compress files, specifically PDF and image formats (and maybe audios or videos as well), while maintaining the original file type and quality. The main goal is to reduce storage space usage on users' devices through efficient, automated optimization. The system identifies the file type using a FileAnalyzer module and applies the most suitable compression method: for images, it uses techniques such as lossless re-encoding, resolution optimization, or metadata removal; for PDFs, it reduces embedded image resolution, removes unused fonts, and optimizes internal structures. The entire process is handled seamlessly within the same file format, ensuring the output remains a valid .pdf or .jpg/.png file. The user interface provides an intuitive experience with options for file selection, output path choice, and progress visualization. Additionally, animations create a friendly and engaging environment ("ISMA File Compressor" welcome and goodbye screens). The project emphasizes efficiency, usability, and clean code structure, offering a practical and lightweight compression solution for everyday users.

2. INTRODUCTION

2.1 Problem Statement

[Describe the specific problem your project addresses and its significance in real-world applications.]

As users increasingly handle large documents and images (and maybe audios or videos as well), storage space and data transfer speed have become significant concerns. Most available compression tools (like ZIP or RAR) create separate archive files that require decompression before use, which can be inconvenient. Our team identified the need for a direct file compressor that optimizes existing files, particularly PDFs and images (and maybe audios or videos as well), without changing their formats. This allows users to save space, share files faster, and retain compatibility with standard viewers. This project focuses on designing an easy-to-use graphical tool that performs on-format compression using efficient optimization algorithms, providing real-world benefits while showcasing key computer science concepts such as file handling, data reduction, and user interface design.

2.2 Project Goals & Objectives

Primary Goal:

[State the main goal of your project]

To design and implement a GUI-based smart file compressor that reduces file size while keeping the same file type (PDF or image (and maybe audios or videos as well)), providing users with an efficient, simple, and user-friendly experience.

Specific Objectives:

1. [Objective 1 - Specific and measurable]
   File Analysis and Detection
   Implement a FileAnalyzer module that automatically identifies whether the selected file is a PDF or an image (e.g., PNG, JPG, PNG, MP3, WAV.....).
   Measurable Outcome: ≥99% accuracy in file detection.

2. [Objective 2 - Specific and measurable]
   PDF Compression
   Apply optimization techniques such as removing redundant objects, downscaling embedded images, and eliminating unused metadata.
   Measurable Outcome: Average size reduction of 40–70% for standard PDFs without visible quality loss.

3. [Objective 3 - Specific and measurable]
   Image Compression
   Implement intelligent image compression using lossless re-encoding, metadata removal, and optional quality-based resizing for JPG and PNG files.
   Measurable Outcome: Achieve at least 50% space reduction while maintaining visual clarity.

4. [Objective 4 - Specific and measurable]

   User Interface and Experience

   Develop a simple and responsive GUI that allows users to easily select files, view compression progress, and choose an output location.

   Include animations and a pleasant color theme to enhance usability and engagement.

   Measurable Outcome: ≥90% user satisfaction during testing.

5. [Objective 5 - Specific and measurable]

   Code Quality and Documentation

   Ensure readable, modular, and maintainable code with extensive inline comments and a detailed README for future improvements.

   Measurable Outcome: 95% code documentation coverage and successful execution across platforms.

2.3 Scope and Limitations

In Scope:

- [Feature 1]
  PDF compression:
  - Reducing embedded image resolution.
  - Removing unused fonts and metadata.
  - Optimizing internal structure using PDF libraries.
- [Feature 2]
  Image compression:
  - Removing EXIF data.
  - Re-encoding with optimized quality.
  - Optional resizing while maintaining format (.jpg, .png).
- [Feature 3]
  Graphical User Interface:
  - File chooser, progress bar, compression summary, and start/exit, animations.

  Batch processing:
  - Allow users to compress multiple files in one session.

  Performance metrics:
  - Display compression ratio and space saved.

Out of Scope:

- [Limitation 1]
  Audio and video files are not supported in this version (may be added later).

- [Limitation 2]
  Lossy compression will be limited to controlled, user-specified quality adjustments.

- [Limitation 3]
  No custom file format (.isma) — output remains the same file type as input.

- [Limitation 4]
  Streaming or live compression is out of scope.

3. SYSTEM DESIGN

3.1 Data Structure Selection & Justification

Data Structure	Purpose	Justification	Time Complexity
Dictionary (Hash Map)	To store file metadata such as file name, size, type, and compression ratio.	Provides quick lookup and update of file information during compression.	O(1) average
Byte Array / Buffer	To handle raw file data during compression and optimization.	Efficient for manipulating file bytes before and after optimization (especially images and PDFs).	O(n) for read/write
Queue (optional)	To manage multiple files waiting to be compressed (batch compression).	Enables sequential processing and progress tracking in the GUI.	O(1) enqueue/dequeue
List / ArrayList	To store references to selected files and output paths.	Easy iteration and access for GUI display and file selection.	O(n)
Tree / Huffman Node (optional)	Could be used in the future for advanced lossless compression.	Extensible design in case we integrate Huffman coding later.	O(n log n) for building tree

3.2 System Architecture

[Describe your system design and include diagrams if available]

Our system follows a **modular architecture**, separating the GUI, core processing, and utility functions.
This allows maintainability, easier debugging, and future scalability (e.g., adding audio or video compression).

Main Components:

1. [Component 1 description]
   GUI Layer (JavaFX or Swing)

- Provides a friendly interface for file selection, compression start, progress bar, and output path.
- Displays compression ratio, before/after file sizes, and a short animation (intro + goodbye).
- Uses **JavaFX** (recommended for modern look) or **Swing** (simpler alternative).

2. [Component 2 description]
   Compressor Engine (Core Layer)

- Handles actual file optimization and compression.
- **For PDFs:** Uses the Apache PDFBox library to reduce size by removing metadata and compressing images.
- **For Images:** Uses javax.imageio and optional TwelveMonkeys ImageIO plugin to reduce quality or re-encode images.
- Works only on the **data portion**, keeping the file format intact (PDF → PDF, JPG → JPG).

3. [Component 3 description]
   File Manager (Utility Layer)

- Detects file type based on **file signature (magic number)** or file extension.
- Can use Files.probeContentType(Path) or manual byte reading via FileInputStream.
- Responsible for reading input files, validating file type, and saving the compressed version in the chosen folder.

4. [Component 4 description]
   Logger & Statistics Module

- Stores information about each compression task (file name, size before/after, time taken).

- Logs data into .txt or .csv for transparency and evaluation of compression performance.

3.3 Class Diagram/System Overview

[Describe your class structure and relationships]

Class: FileManager

Attributes:

- File file
- String fileType
- long fileSize

Methods:

- String detectFileType(File file) → Uses Files.probeContentType() or byte reading.
- FileInfo getFileInfo(File file) → Returns name, type, and size.
- void saveCompressedFile(File source, File destination) → Saves optimized file.

Class: Compressor

Attributes:

- double compressionRatio
- String algorithm (e.g., “PDFBox”, “ImageIO”)

Methods:

- File compressPDF(File pdfFile) → Uses **Apache PDFBox** to optimize.
- File compressImage(File imageFile, float quality) → Uses **ImageIO** to re-encode.
- double calculateRatio(long originalSize, long newSize) → Returns reduction %.

Class: GUIManager (JavaFX Controller)

Attributes:

- List<File> selectedFiles
- ProgressBar progressBar
- Label statusLabel

Methods:

- void chooseFiles() → Opens file chooser dialog.
- void startCompression() → Sends files to Compressor.
- void showSummary(FileInfo info) → Displays before/after stats and goodbye message.

Class: Logger

Attributes:

- List<String> entries
- File logFile

Methods:

- void logActivity(FileInfo info) → Saves a summary of each operation.
- void exportLogs() → Writes logs to disk as CSV/TXT.

Class: FileInfo

Attributes:

- String fileName
- String fileType
- long originalSize
- long newSize
- double ratio

Methods:

- Getters/Setters
- toString() → For formatted output/logging.

Summary

The **Java-based system** is modular, efficient, and practical.
It integrates **JavaFX** for GUI, **Apache PDFBox** for PDF compression, and **ImageIO** for image optimization.
This architecture ensures:

- Real-time compression feedback
- Same-file-type output
- Smooth, maintainable, and extendable codebase