

Project Documentation (File Compression Utility)

1. Project Overview

The File Compression Utility is a Java-based desktop application that allows users to compress and decompress files using different compression algorithms (ZIP and RAR). The application features a graphical user interface (GUI) built with Java Swing and implements multiple design patterns to ensure maintainability, extensibility, and code reusability.

2. Key Features

- Support for multiple compression formats (ZIP, RAR)
- Single and multi-file compression
- File type-specific preprocessing (Text, Image, Video)
- User-friendly GUI with progress tracking
- Extensible architecture using design patterns

3. Design Patterns Used

The project implements five major design patterns:

#	Pattern	File	Purpose
1	Singleton	CompressionManager.java FileHandler.java	Ensures only ONE instance exists, controls access to compression operations and file handling through a single global point.
2	Factory	CompressionFactory.java, File TypeFactory.java	Creates objects without exposing creation logic, returns the right interface (Compressor or FileTypeProcessor) based on input type.
3	Adapter	RarCompressorAdapter.java	Wraps an incompatible interface (WinRAR command-line tool) to work with the Compressor interface, bridges external software with your code.
4	Prototype	FileSelection.java	Allows cloning/copying objects. Users can save a file selection and duplicate it with clone() to create variations quickly.
5	Builder	ArchiveNameBuilder.java	Constructs complex objects step-by-step. Builds output archive filenames with optional parts (prefix, timestamp, extension).

4. Class Descriptions

interfaces

#	Class	Description
1	Compressor	interface defining compression operations: compress(), decompress(), compressMultiple(), and getExtension()
2	FileTypeProcessor	Interface for file preprocessing before compression: prepareForCompression(), getFileType(), getProcessingDescription()

5. Factory Pattern Classes

#	Class	Description
1	CompressionFacotry	Creates appropriate Compressor instances based on type ("ZIP" or "RAR"). Returns ZipCompressor or RarCompressorAdapter Classes
2	FileTypeFactory	Creates appropriate FileTypeProcessor based on file extension. Maps extensions to Text, Image, Video, or Default processors

6. Singleton Pattern Classes

#	Class	Description
1	CompressionManager	Central manager for compression/decompression operations. Uses lazy initialization. Provides progress bar integration and multi-file compression
2	FileHandler	Singleton for file I/O operations: readFile(), writeFile(), fileExists(). Uses lazy initialization

7. Adapter Pattern Class

#	Class	Description
1	RarComressorAdapter	Adapts external WinRAR command-line tools (rar.exe/unrar.exe) to work with the Compressor interface. Translates method calls to command-line executions

8.Builder Pattern Class

#	Class	Description
1	ArchiveNameBuilder	Constructs customized archive filenames step-by-step. Supports: withPrefix(), withTimestamp(), withSuffix(), withExtension(), withSeparator(). Example: "backup_myfile_20231210_143000.zip"

9.Prototype Pattern Class

#	Class	Description
1	FileSelection	Stores file selection state (files, compression type, output directory). Implements Cloneable to create deep copies. Allows users to duplicate and modify selections

10.Compressor Implementations

#	Class	Description
1	ZipCompressor	Native Java ZIP compression using java.util.zip. Handles single and multi-file compression/decompression without external dependencies

11.FileTypeProcessor Implementations

#	Class	Description
1	TextProcessor	Optimizes text files for compression: normalizes line endings, removes trailing whitespace, reduces blank lines
2	ImageProcessor	Processes images: extracts metadata (dimensions), validates format. Returns original data as images are already compressed
3	VideoProcessor	Processes videos: detects format via magic numbers (MP4, AVI, MKV), logs file info. Returns original data as videos are already compressed
4	DefaultProcessor	Handles unknown file types with no preprocessing - passes data through unchanged