

LAPORAN
TUGAS PROYEK
**“Optimalisasi Kompresi File Dengan CompactJV:
Sebuah Pendekatan Berbasis Java untuk Kompresi
File Software dan Game”**

Diajukan untuk memenuhi salah satu praktikum Mata Kuliah Pemrograman Berorientasi
Objek yang di ampu oleh:

Faisal Fajri Rahani, S.Si., M.Cs. Teknik Informatika



Disusun Oleh:

Mohammad Farid Hendianto 2200018401

Reyhanssan Islamey 2200018411

Rendie Abdie Saputra 2200018094

Fadhil Raifan Andika 2200018458

PROGRAM STUDI INFORMATIKA
UNIVERSITAS AHMAD DAHLAN
FAKULTAS TEKNOLOGI INDUSTRI
TAHUN 2024

KATA PENGANTAR

Dengan merendahkan hati, kami panjatkan puji dan syukur kepada Tuhan Yang Maha Esa, Allah SWT, yang telah memberikan rahmat, hidayah, dan karunia-Nya sehingga kami dapat menyelesaikan laporan proyek akhir ini. Laporan ini bertajuk "Optimalisasi Kompresi File Dengan CompactJV: Sebuah Pendekatan Berbasis Java untuk Kompresi File Software dan Game". Kami berterima kasih atas berbagai pelajaran yang kami peroleh selama proses penyelesaian proyek ini.

Proses penulisan laporan ini merupakan perjalanan yang panjang dan penuh tantangan, namun juga membawa banyak pengetahuan dan pengalaman berharga. Kami berusaha sebaik mungkin untuk mengaplikasikan dan mengimplementasikan konsep-konsep pemrograman berorientasi objek yang telah kami pelajari selama ini dalam proyek ini.

Ucapan terima kasih yang tak terhingga kami sampaikan kepada Bapak Faisal Fajri Rahani, S.Si., M.Cs., dosen Teknik Informatika Universitas Ahmad Dahlan. Beliau telah memberikan bimbingan, arahan, dan dorongan yang sangat berarti dalam proses pengerjaan proyek ini. Tanpa bimbingan dan arahan beliau, kami tidak mungkin bisa menyelesaikan proyek ini dengan hasil yang memuaskan.

Laporan ini tentunya masih memiliki banyak kekurangan. Oleh karena itu, kami sangat mengharapkan kritik dan saran dari semua pihak yang membaca laporan ini. Kritik dan saran yang konstruktif akan sangat membantu kami untuk terus belajar dan berkembang, serta memperbaiki kualitas laporan kami di masa mendatang.

Akhirnya, kami berharap semoga laporan ini dapat memberikan manfaat dan pengetahuan baru bagi kita semua. Semoga laporan ini dapat membantu kita semua dalam memahami lebih dalam tentang cara kerja dan implementasi unit pemrosesan dalam komputer, serta memberikan inspirasi untuk terus berinovasi dan berkembang dalam bidang teknologi.

Yogyakarta, Januari 2024

Ketua Kelompok



Mohammad Farid Hendianto

DAFTAR ISI

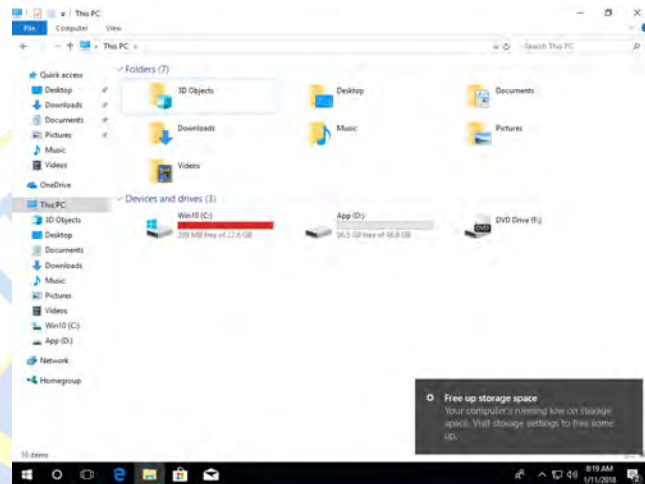
KATA PENGANTAR.....	ii
DAFTAR ISI.....	iii
BAB I PENDAHULUAN.....	5
1.1. Latar Belakang Masalah	5
1.2. Rumusan Masalah	6
1.3. Tujuan Pembuatan Aplikasi	6
1.4. Manfaat Pembuatan Aplikasi	6
BAB II PEMBAHASAN.....	7
2.1. Tentang Aplikasi	7
2.2. Flowchart	8
2.2.1. Main Flowchart	9
2.2.2. Class UI Flowchart.....	24
2.3. Use Case Diagram	44
2.4. UML Class Diagram	47
2.4.1. Class Application.....	48
2.4.2. Class Controller.....	48
2.4.3. Class ExecutorServiceManager	50
2.4.4. Class File	51
2.4.5. Class Compressor.....	52
2.4.6. Class Size.....	52
2.4.7. Class ButtonHandler.....	53
2.4.8. Class LoadingBarUI	54
2.4.9. Class FilePathUI.....	55
2.4.10. Interface InformationUI.....	55
2.4.11. Class CPUInformationUI dan MemoryInformationUI.....	56
2.5. Design Interface	57
2.5.1. Deskripsi Umum.....	58
2.5.2. Riwayat Design CompactJV	58
2.5.3. Komponen Utama	63
2.6. Code.....	64
2.6.1. Class Button Handler.....	65
2.6.2. Class Button UI	66

2.6.3.	Class CPU Information	68
2.6.5.	Class Disk Information.....	71
2.6.6.	Class File Path UI.....	73
2.6.7.	Class Information UI	76
2.6.8.	Class Loading bar	77
2.6.9.	Class Memory Information	78
2.6.10.	Class Navbar UI	79
2.6.11.	Class Tool Tip UI	81
2.6.12.	Class Windows Controller UI	83
2.6.13.	Class Command Runner	85
2.6.14.	Class Compressor.....	87
2.6.15.	Class Controller.....	88
2.6.16.	Class Disk.....	89
2.6.17.	Class Executor Service Manager	91
2.6.18.	Class File.....	93
2.6.19.	Class Size.....	94
BAB III PENUTUP.....		98
3.1.	Kesimpulan.....	98
3.2.	Kritik dan Saran	98
DAFTAR GAMBAR.....		99

BAB I

PENDAHULUAN

1.1. Latar Belakang Masalah



Gambar 1 Ilustrasi kepenuhan storage.

Di era digital yang semakin maju ini, pertukaran data menjadi kegiatan sehari-hari dalam berbagai bidang kehidupan. Dalam dunia pendidikan, data berupa file dokumen, presentasi, dan video ajar digunakan untuk mendukung proses belajar mengajar. Di bidang bisnis, pertukaran data berupa laporan keuangan, proposal, dan lain sebagainya menjadi hal yang rutin dilakukan. Sementara itu, dalam bidang hiburan, file data berupa software dan game menjadi komoditas yang banyak dicari.

Sayangnya, ukuran file untuk data-data tersebut seringkali besar, terutama untuk software dan game. File dengan ukuran besar memerlukan waktu yang lama untuk ditransfer dan memakan banyak ruang penyimpanan. Hal ini tentunya menjadi kendala, terutama jika koneksi internet yang digunakan memiliki bandwidth terbatas atau ruang penyimpanan yang tersedia sudah hampir penuh.

Untuk mengatasi masalah tersebut, diperlukan sebuah metode yang dapat mengurangi ukuran file tanpa mengurangi kualitas atau fungsi dari file tersebut. Salah satu metode yang dapat digunakan adalah kompresi file. Kompresi file adalah proses pengurangan ukuran file dengan cara mengubah representasi data menjadi bentuk yang lebih efisien.

Namun, proses kompresi file seringkali memerlukan pengetahuan teknis dan waktu yang cukup lama. Oleh karena itu, diperlukan sebuah alat yang dapat memudahkan dan mengoptimalkan proses kompresi file. CompactJV adalah alat kompresi yang dibuat dengan menggunakan Java dan dirancang khusus untuk pengguna sistem operasi Windows.

CompactJV mendukung beberapa algoritma kompresi, termasuk XPRESS4K, XPRESS8K, XPRESS16K, XPRESS, dan LZX. Pengguna dapat memilih algoritma yang paling sesuai dengan kebutuhan mereka. Selain itu, CompactJV juga memiliki antarmuka

pengguna yang ramah dan mudah digunakan, sehingga pengguna dapat melakukan kompresi file dengan mudah dan cepat.

Selain itu, CompactJV juga dilengkapi dengan fitur-fitur yang mendukung proses kompresi file, seperti fitur pemilihan algoritma kompresi, fitur penyesuaian tingkat kompresi, dan fitur preview hasil kompresi. Dengan fitur-fitur ini, pengguna dapat melakukan kompresi file dengan lebih mudah dan efisien.

1.2. Rumusan Masalah

Berdasarkan latar belakang masalah di atas, dapat dirumuskan beberapa masalah sebagai berikut:

- Bagaimana cara mengurangi ukuran file tanpa mengurangi kualitas dan fungsi file tersebut?
- Bagaimana cara mengoptimalkan proses kompresi file untuk software dan game?
- Bagaimana cara membuat alat kompresi yang mudah digunakan dan efisien?

1.3. Tujuan Pembuatan Aplikasi

Berikut ini adalah tujuan dari pembuatan CompactJV:

- Mengembangkan alat kompresi file yang efisien dan mudah digunakan.
- Mengoptimalkan proses kompresi file, khususnya untuk software dan game.
- Mengurangi ukuran file tanpa mengurangi kualitas dan fungsi file tersebut.

1.4. Manfaat Pembuatan Aplikasi

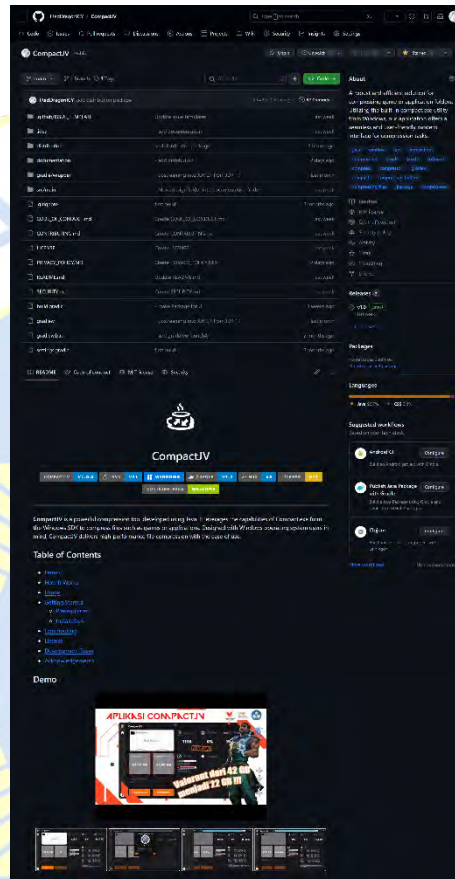
Berikut ini adalah manfaat yang dapat diperoleh dari pembuatan CompactJV:

- Pengguna dapat menghemat ruang penyimpanan dengan mengurangi ukuran file.
- Pengguna dapat menghemat waktu dalam transfer file karena ukuran file yang lebih kecil.
- Pengguna dapat memilih algoritma kompresi yang sesuai dengan kebutuhan mereka.
- Pengguna dapat menggunakannya dengan mudah berkat antarmuka pengguna yang ramah.
- Pengguna dapat menyesuaikan tingkat kompresi sesuai dengan kebutuhan mereka.
- Pengguna dapat melihat preview hasil kompresi sebelum melakukan proses kompresi.

BAB II

PEMBAHASAN

2.1. Tentang Aplikasi



Gambar 2 Aplikasi di distribusikan di Github

Aplikasi **CompactJV** adalah alat kompresi yang kuat yang dikembangkan menggunakan Java. Aplikasi ini memanfaatkan kemampuan Compact.exe dari Windows SDK untuk mengkompresi file seperti game atau aplikasi. Dirancang dengan mempertimbangkan pengguna sistem operasi Windows, CompactJV memberikan kompresi file berkinerja tinggi dengan kemudahan penggunaan.

CompactJV menggunakan alat Compact.exe dari Windows SDK untuk mengompresi file. Aplikasi ini mendukung beberapa algoritma kompresi, termasuk XPRESS4K, XPRESS8K, XPRESS16K, XPRESS, dan LZX. Berikut adalah gambaran singkat tentang bagaimana setiap algoritma bekerja:

1. **XPRESS4K:** Ini adalah algoritma kompresi yang cepat yang memberikan rasio kompresi yang wajar. '4K' dalam namanya merujuk pada ukuran jendela geser yang digunakan selama kompresi.

2. **XPRESS8K dan XPRESS16K:** Ini adalah variasi dari algoritma XPRESS4K. Mereka menggunakan jendela geser yang lebih besar 8K dan 16K, masing-masing. Hal ini memungkinkan mereka untuk mencapai rasio kompresi yang lebih baik, tetapi dengan biaya kinerja yang lebih lambat.
3. **XPRESS:** Algoritma ini memberikan keseimbangan antara kinerja dan rasio kompresi. Ini lebih efisien daripada XPRESS4K, XPRESS8K, dan XPRESS16K, tetapi tidak sekuat LZX.
4. **LZX:** Ini adalah algoritma kompresi berkinerja sangat tinggi. Itu memberikan rasio kompresi yang sangat baik, tetapi juga cukup lambat dibandingkan dengan algoritma lainnya.

Saat memilih untuk mengompresi file dengan CompactJV, program tersebut menggunakan salah satu algoritma ini untuk mengurangi ukuran file tersebut. Algoritma spesifik yang digunakan bergantung pada opsi yang Anda pilih.

Berikut cara menggunakan CompactJV:

1. Pilih file yang ingin Anda kompres.
2. Pilih algoritma kompresi yang ingin Anda gunakan.
3. Klik tombol 'Compress'.

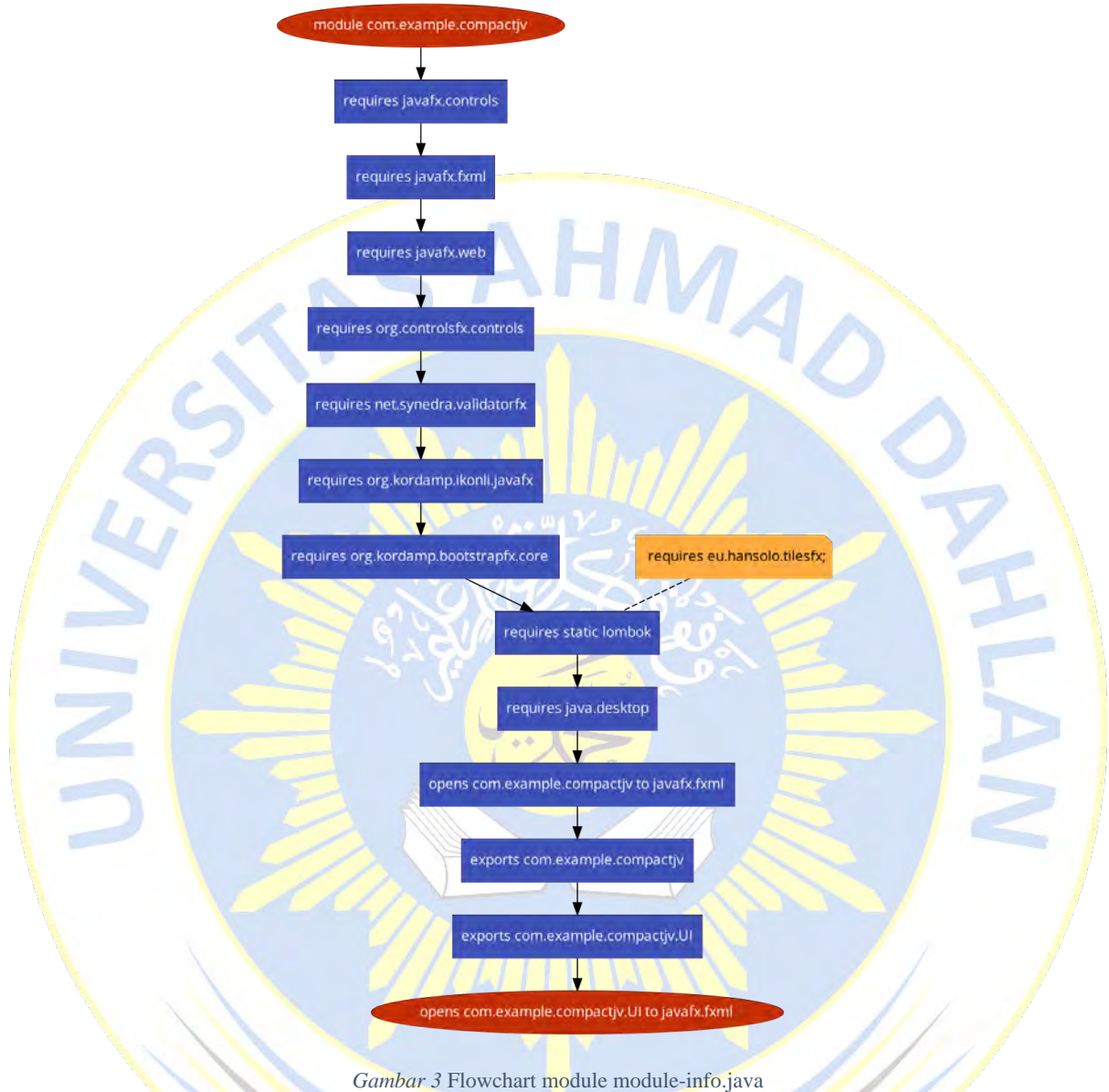
CompactJV kemudian akan mengompresi file menggunakan algoritma yang dipilih. File yang dikompresi akan disimpan di lokasi yang sama dengan file aslinya.

Untuk informasi lebih lanjut dapat melihat link github berikut:

<https://github.com/IRedDragonICY/CompactJV>

2.2. Flowchart

2.2.1. Main Flowchart



Gambar 3 Flowchart module module-info.java

Modul CompactJV terdiri dari beberapa dependensi yang dibutuhkan untuk menjalankan aplikasi. Pertama-tama, kita memerlukan pustaka JavaFX untuk antarmuka pengguna dan pemrosesan XML. Selain itu, ada beberapa dependensi lain seperti ControlsFX, ValidatorFX, Ikonli, dan BootstrapFX yang digunakan untuk menambahkan fitur-fitur tambahan dan meningkatkan tampilan antarmuka pengguna.

Aplikasi ini juga memerlukan Lombok untuk mengurangi boilerplate code dengan menggunakan anotasi, dan Java Desktop module untuk integrasi dengan desktop environment.

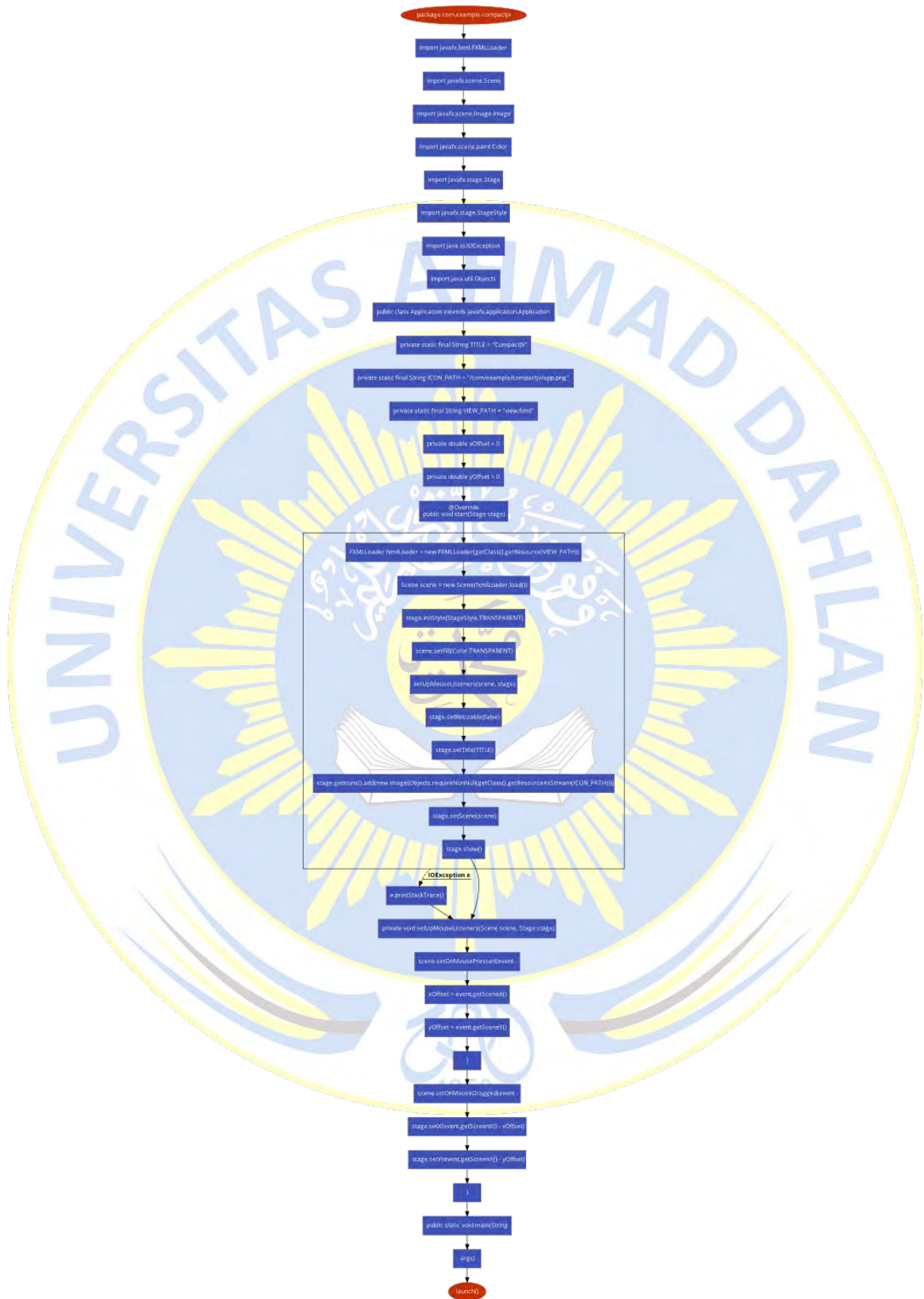
Selanjutnya, kita memastikan bahwa paket-paket yang dibutuhkan dapat diakses dan diekspor dengan benar. Paket `com.example.compactjv` diakses oleh JavaFX untuk loading FXML, sementara paket `com.example.compactjv.UI` juga diakses untuk antarmuka pengguna.

Dalam eksekusi program, modul ini membuka paket-paket tersebut ke FXML untuk dapat diakses oleh JavaFX. Kemudian, aplikasi diexport untuk digunakan.

Penting untuk diingat bahwa struktur modul ini dirancang untuk mencapai keterbacaan dan pemeliharaan yang baik. Setiap kebutuhan dan dependensi diatur dengan rapih untuk memastikan bahwa aplikasi dapat berjalan dengan baik dan mudah dikembangkan.

Contoh eksekusi program ini dimulai dengan memeriksa ketersediaan dependensi yang diperlukan, seperti JavaFX, ControlsFX, dan lainnya. Setelah itu, aplikasi CompactJV akan dijalankan dengan mengeksekusi modul utama. Aplikasi ini kemudian memuat antarmuka pengguna dengan menggunakan FXML dan menampilkan halaman utama. Pengguna dapat melakukan berbagai tindakan, seperti memilih file yang akan diompres, memilih algoritma kompresi yang diinginkan, dan menekan tombol "Compress". Setelah itu, program akan menggunakan Compact.exe dari Windows SDK untuk melakukan kompresi sesuai dengan algoritma yang dipilih. Hasilnya akan disimpan di lokasi yang sama dengan file asli.

Proses ini memungkinkan pengguna untuk menggunakan CompactJV dengan mudah dan efektif, sambil memanfaatkan fitur-fitur kompresi yang disediakan.



Gambar 4 Flowchart Class Application.java

Flowchart di atas adalah implementasi dari kelas `Application` pada aplikasi CompactJV. Mari kita bahas alur programnya secara detail.

Pertama-tama, aplikasi diinisialisasi dengan menetapkan beberapa konstanta seperti judul (`TITLE`), path ikon (`ICON_PATH`), dan path tampilan (`VIEW_PATH`). Selain itu, terdapat variabel `xOffset` dan `yOffset` yang digunakan untuk menyimpan posisi awal saat mouse ditekan.

Kemudian, metode `start` diimplementasikan. Pada metode ini, objek `FXMLLoader` digunakan untuk memuat tampilan dari berkas FXML yang diidentifikasi oleh `VIEW_PATH`. Selanjutnya, dibuat objek `Scene` yang diinisialisasi dengan tampilan yang dimuat.

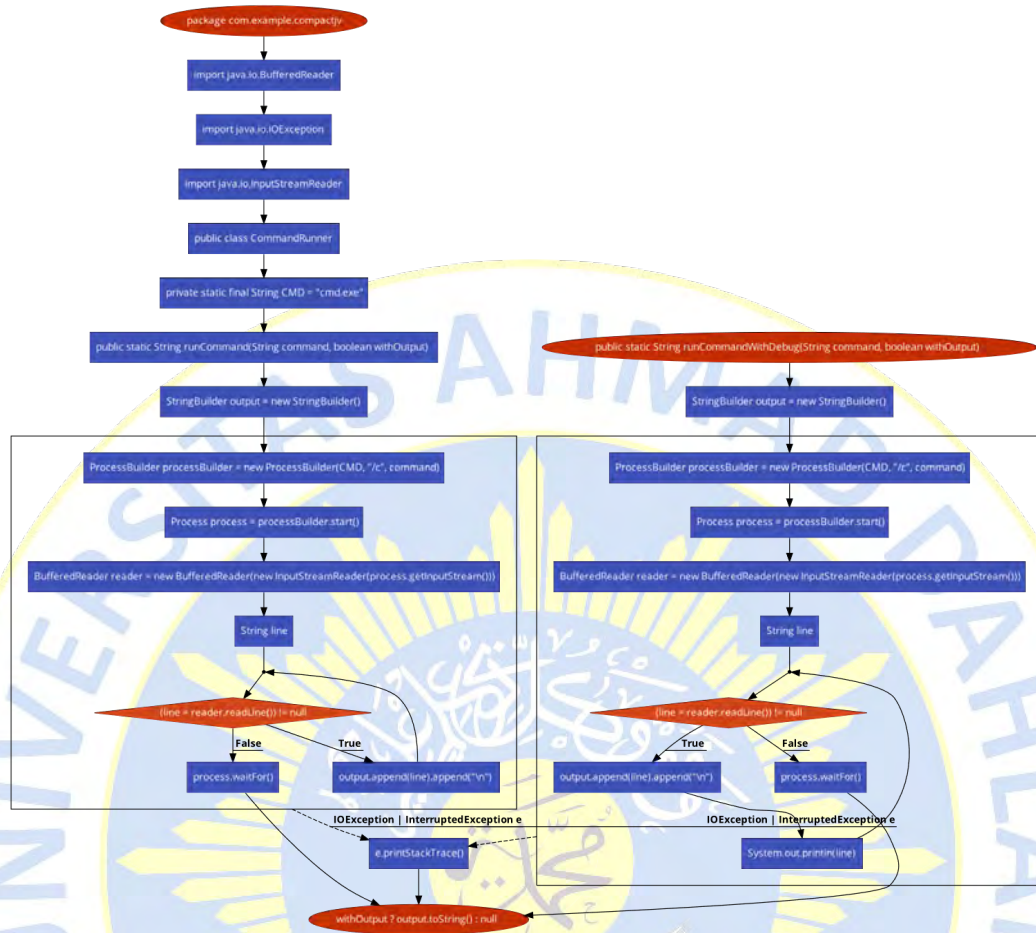
Stage diatur dengan gaya `StageStyle.TRANSPARENT`, sehingga tampilan aplikasi dapat dibuat transparan. Selain itu, properti lain seperti `resizable`, `title`, `icons`, dan `scene` diatur sesuai kebutuhan.

Metode `setUpMouseListeners` ditambahkan untuk menangani peristiwa mouse saat pengguna menekan dan menarik mouse. Ini memungkinkan pengguna untuk menggeser jendela aplikasi tanpa batas yang telah ditetapkan.

Terakhir, metode `main` diimplementasikan dengan memanggil `launch()`. Metode ini merupakan titik masuk utama untuk menjalankan aplikasi JavaFX.

Eksekusi program dimulai dengan memanggil metode `main`. Selanjutnya, metode `start` dijalankan, dan tampilan aplikasi ditampilkan. Pengguna dapat memilih file yang akan diompres, memilih algoritma kompresi, dan menekan tombol "Compress" sesuai dengan flowchart aplikasi yang telah diimplementasikan.

Selama pengguna menggunakan aplikasi, mereka dapat menggeser jendela aplikasi ke posisi yang diinginkan berkat penanganan mouse yang telah diimplementasikan. Keseluruhan, flowchart ini memberikan gambaran tentang bagaimana aplikasi CompactJV dijalankan dan berinteraksi dengan pengguna.



Gambar 5 Flowchart Class CommandRunner.java

Flowchart di atas adalah implementasi dari kelas `CommandRunner` pada aplikasi CompactJV. Mari kita bahas alur programnya secara detail.

Pertama-tama, kelas `CommandRunner` memiliki dua metode utama, yaitu `runCommand` dan `runCommandWithDebug`. Kedua metode ini digunakan untuk menjalankan perintah pada Command Prompt.

Metode `runCommand` digunakan untuk menjalankan perintah tanpa menampilkan output secara langsung pada console. Metode ini menerima dua parameter, yaitu `command` yang merupakan perintah yang akan dijalankan, dan `withOutput` yang menentukan apakah output dari perintah tersebut akan dikembalikan atau tidak.

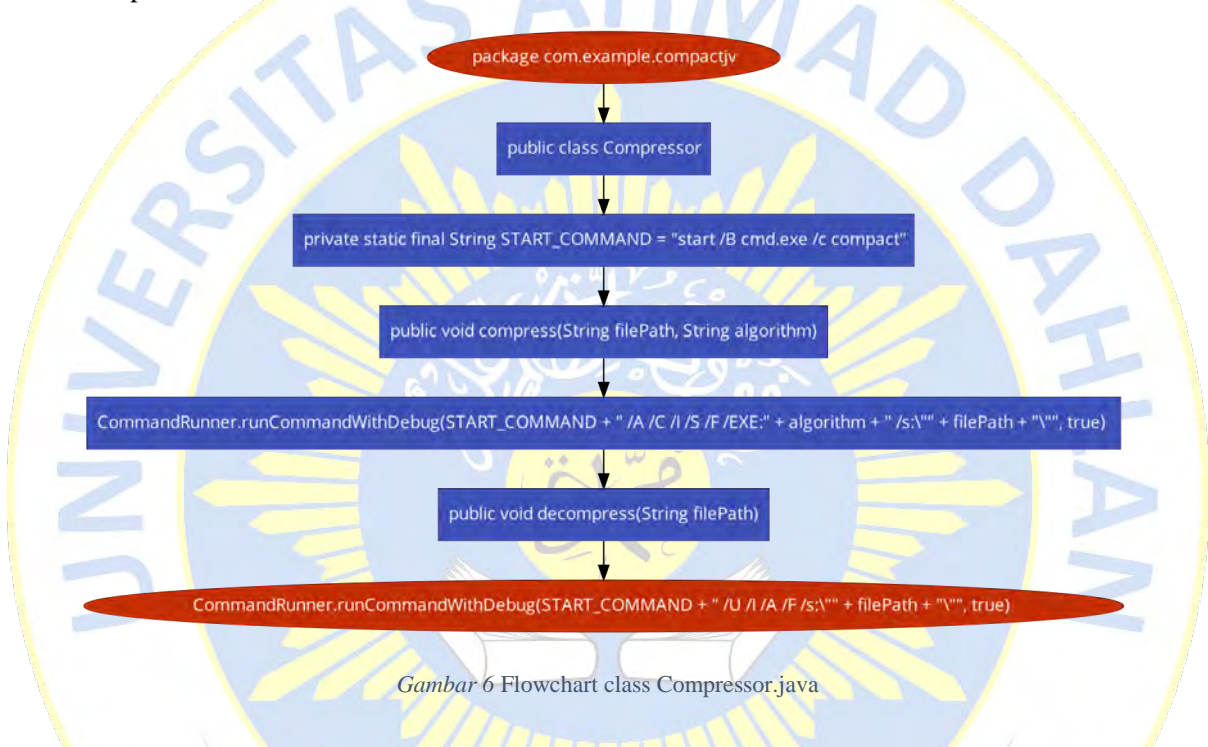
Selanjutnya, sebuah `ProcessBuilder` dibuat untuk membangun proses yang akan menjalankan perintah. Perintah tersebut dijalankan dengan menggunakan `cmd.exe /c`. Proses tersebut kemudian membaca output dari perintah yang dijalankan dan mengumpulkannya ke dalam `StringBuilder`. Proses menunggu sampai perintah selesai dijalankan dengan menggunakan `process.waitFor()`.

Metode `runCommandWithDebug` memiliki fungsi yang sama dengan `runCommand`, namun dengan tambahan fungsi debug. Selain mengumpulkan output ke

dalam ``StringBuilder``, output juga ditampilkan pada console menggunakan ``System.out.println(line)``. Hal ini berguna untuk melihat detail eksekusi perintah pada console selama debug.

Kedua metode ini dapat digunakan untuk menjalankan perintah eksternal, seperti menggunakan `Compact.exe` dari Windows SDK untuk melakukan kompresi file, sesuai dengan algoritma yang dipilih oleh pengguna pada aplikasi `CompactJV`.

Contoh penggunaan kelas ``CommandRunner`` dapat ditemukan dalam berbagai bagian aplikasi `CompactJV`, terutama pada bagian penggunaan `Compact.exe` untuk kompresi file.



Gambar 6 Flowchart class `Compressor.java`

Flowchart di atas adalah implementasi dari kelas `Compressor` pada aplikasi `CompactJV`. Mari kita bahas alur programnya secara detail.

Kelas `Compressor` memiliki dua metode utama, yaitu `compress` dan `decompress`, yang digunakan untuk melakukan kompresi dan dekompresi file menggunakan utilitas `Compact.exe` dari Windows SDK.

Metode `compress` menerima dua parameter, yaitu `filePath` yang merupakan path dari file yang akan dikompresi, dan `algorithm` yang merupakan algoritma kompresi yang dipilih oleh pengguna. Algoritma ini kemudian digunakan sebagai parameter pada perintah `Compact.exe`.

Pertama-tama, metode `compress` menggunakan kelas `CommandRunner` untuk menjalankan perintah `Compact.exe` dengan opsi yang sesuai. Perintah tersebut dijalankan secara asynchronous menggunakan `start /B` untuk mencegah jendela Command Prompt muncul selama proses kompresi. Opsi `/A /C /I /S /F` digunakan untuk memastikan bahwa

seluruh folder dan file di dalamnya ikut terkompresi. Opsi /EXE: digunakan untuk menentukan algoritma kompresi yang dipilih oleh pengguna.

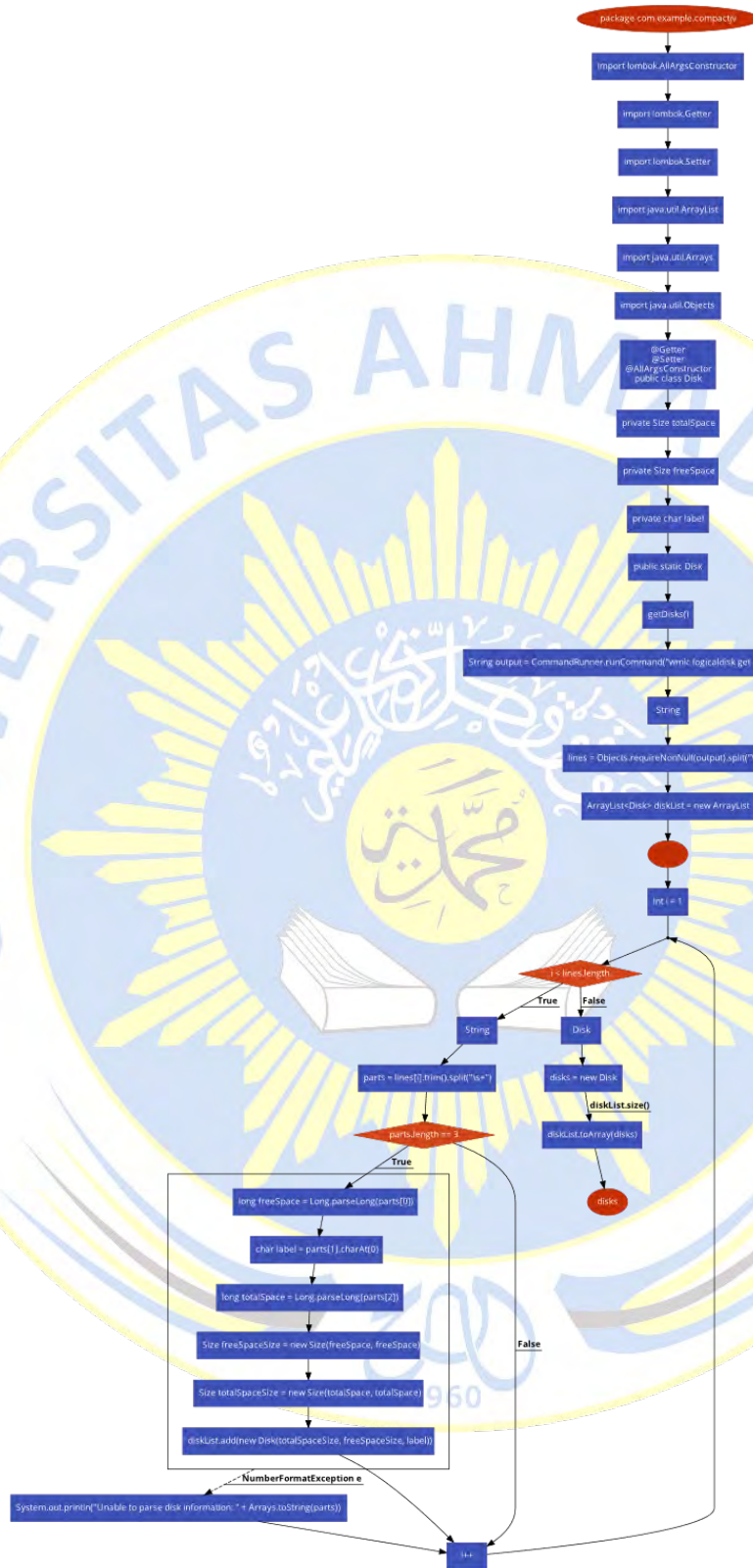
Proses kompresi dilakukan pada path yang diberikan oleh pengguna (filePath). Output dari proses ini ditampilkan pada console untuk keperluan debug dengan menggunakan metode runCommandWithDebug dari kelas CommandRunner.

Metode decompress digunakan untuk melakukan dekompresi pada file yang telah dikompresi sebelumnya. Mirip dengan metode compress, metode decompress juga menggunakan CommandRunner untuk menjalankan perintah Compact.exe dengan opsi yang sesuai. Opsi /U digunakan untuk menunjukkan bahwa proses ini adalah proses dekompresi.

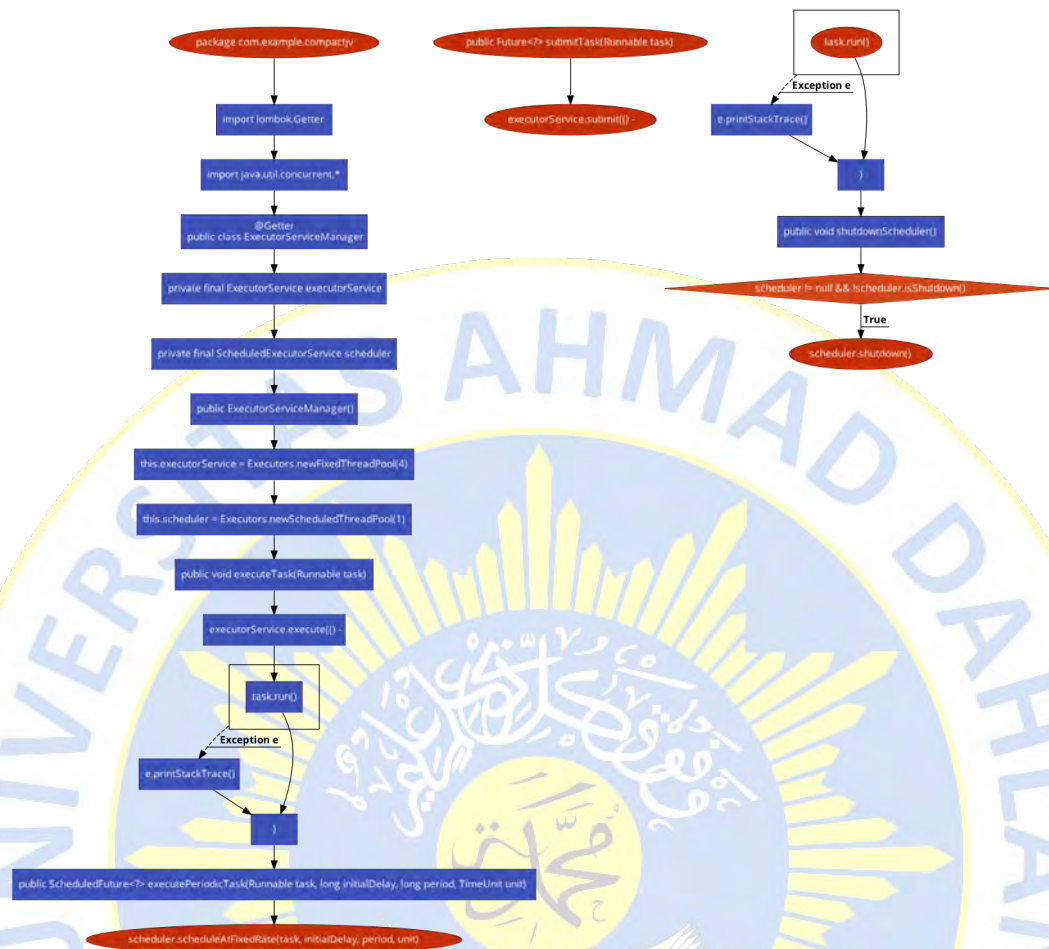
Pada kedua metode, output dari perintah Compact.exe ditampilkan pada console selama proses debug. Dengan ini, pengembang dapat memantau eksekusi perintah Compact.exe secara langsung.

Pengguna dapat menggunakan kelas Compressor ini untuk melakukan operasi kompresi dan dekompresi pada file dengan mudah melalui antarmuka pengguna CompactJV.

Contoh penggunaan kelas Compressor dapat ditemukan dalam berbagai bagian aplikasi CompactJV, terutama pada bagian yang terkait dengan penggunaan Compact.exe untuk mengelola kompresi dan dekompresi file.



Gambar 7 Flowchart class Disk.java



Gambar 8 Flowchart class ExecutorServiceManager.java

Flowchart di atas menggambarkan pengelolaan ExecutorService dalam aplikasi CompactJV.

Pertama-tama, sebuah objek ExecutorServiceManager dibuat dengan menggunakan konstruktor. Objek ini memiliki dua buah ExecutorService: executorService dan scheduler. executorService digunakan untuk mengeksekusi tugas-tugas secara asinkron, sementara scheduler digunakan untuk mengeksekusi tugas-tugas secara periodik.

Setelah objek ExecutorServiceManager dibuat, pengguna dapat menggunakan metode-metode yang disediakan untuk mengeksekusi tugas-tugas.

Metode pertama adalah executeTask, yang menerima sebuah Runnable task sebagai parameter. Metode ini mengeksekusi tugas tersebut menggunakan executorService. Dalam proses eksekusi, apabila terjadi Exception, pesan kesalahan akan dicetak ke konsol.

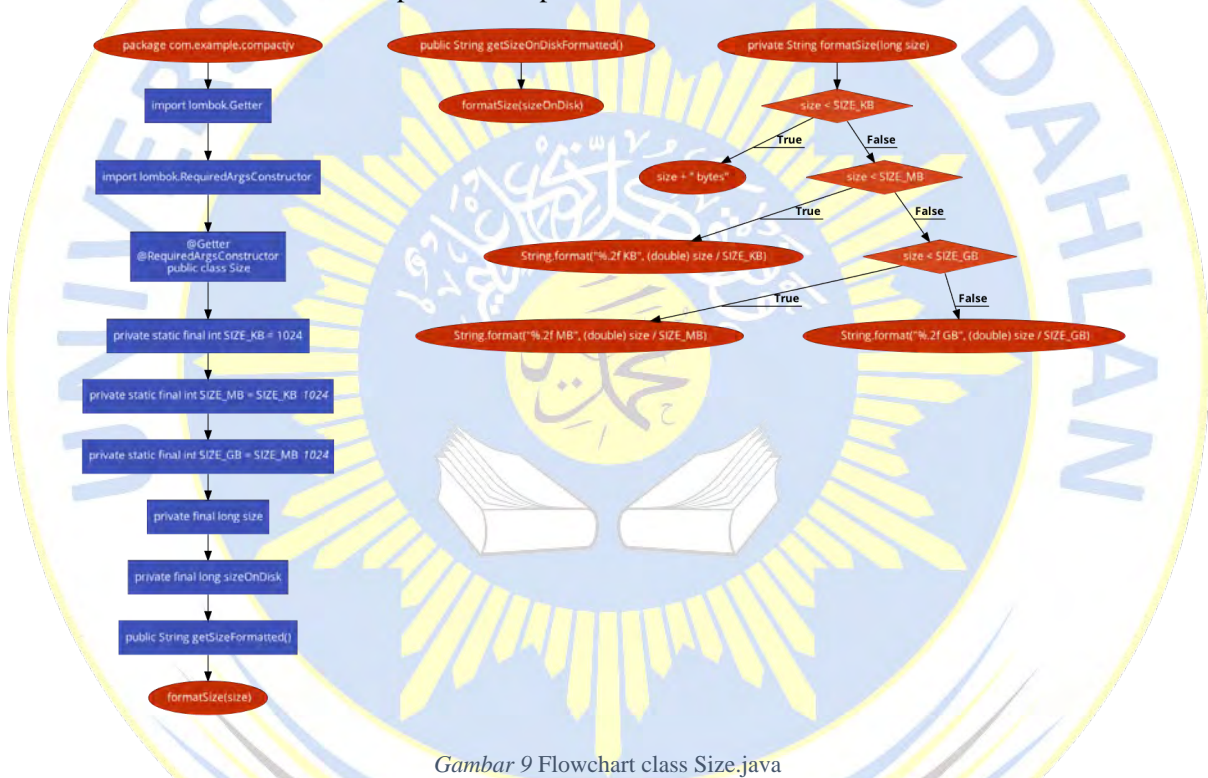
Metode kedua adalah executePeriodicTask, yang digunakan untuk mengeksekusi tugas secara periodik. Metode ini menerima tiga parameter tambahan: initialDelay

(penundaan awal), period (interval antara eksekusi berurutan), dan unit (satuan waktu). Metode ini menggunakan scheduler untuk menjadwalkan dan mengeksekusi tugas secara periodik.

Metode ketiga adalah submitTask, yang mirip dengan executeTask namun mengembalikan objek Future. Objek Future ini dapat digunakan untuk melacak status dan hasil dari eksekusi tugas.

Terakhir, terdapat metode shutdownScheduler yang digunakan untuk menonaktifkan scheduler jika belum dinonaktifkan. Hal ini berguna saat aplikasi CompactJV ditutup atau tidak memerlukan jadwal tugas periodik lagi.

Secara umum, flowchart ini menyediakan pengelolaan eksekusi tugas dengan dua jenis layanan eksekusi yang berbeda, yaitu executorService dan scheduler, untuk menjawab kebutuhan konkret dalam aplikasi CompactJV.



Gambar 9 Flowchart class Size.java

Flowchart di atas menggambarkan kelas `Size` yang digunakan untuk mengelola dan memformat ukuran file.

Pertama-tama, sebuah objek `Size` dibuat dengan dua parameter konstruktor: `size` (ukuran file) dan `sizeOnDisk` (ukuran file di disk). Objek ini juga memiliki konstanta untuk satuan ukuran, yaitu `SIZE_KB`, `SIZE_MB`, dan `SIZE_GB`.

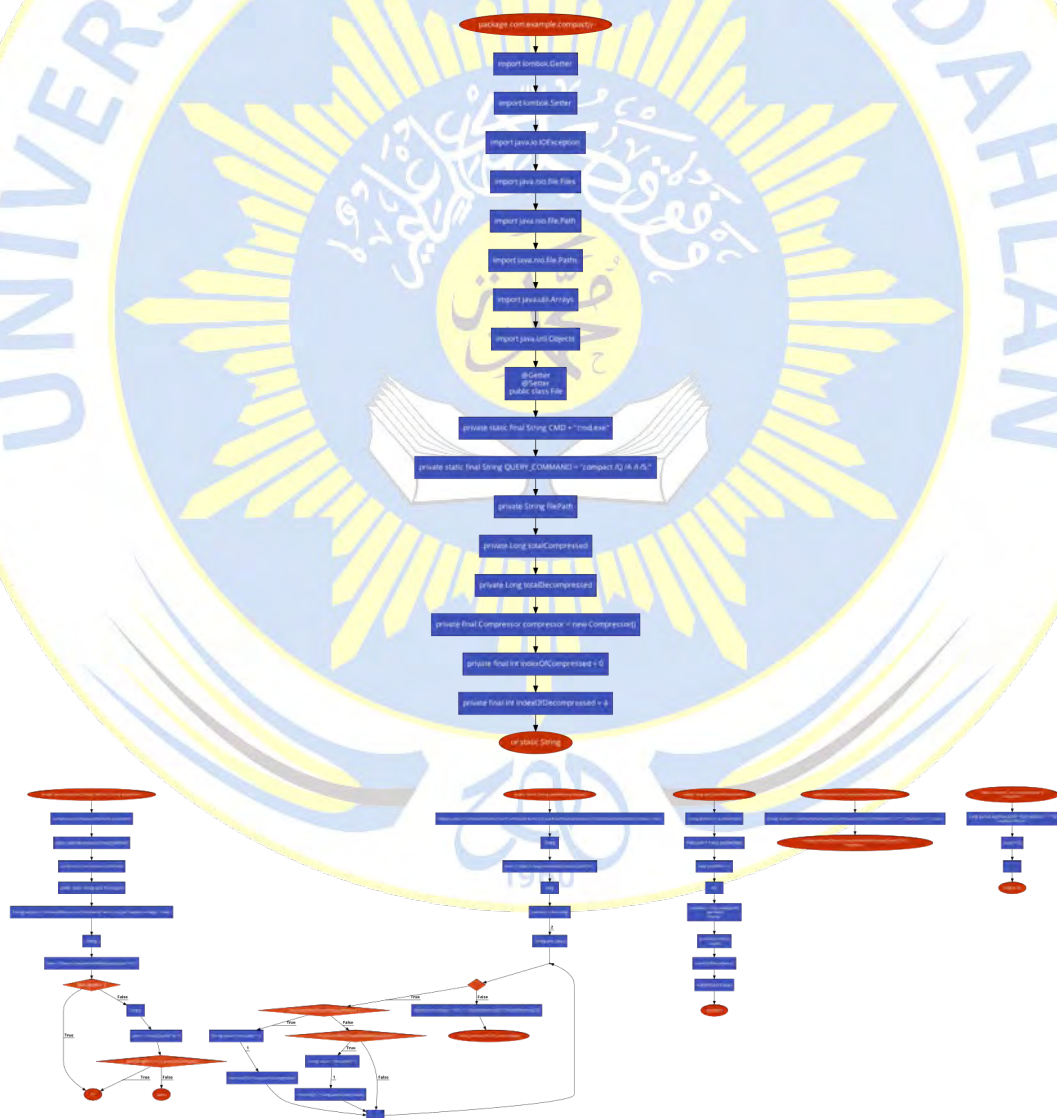
Selanjutnya, terdapat dua metode publik yang dapat dipanggil dari luar kelas, yaitu `getSizeFormatted` dan `getSizeOnDiskFormatted`. Metode pertama mengembalikan ukuran file yang telah diformat sebagai string menggunakan metode privat `formatSize`. Metode kedua melakukan hal yang sama untuk ukuran file di disk.

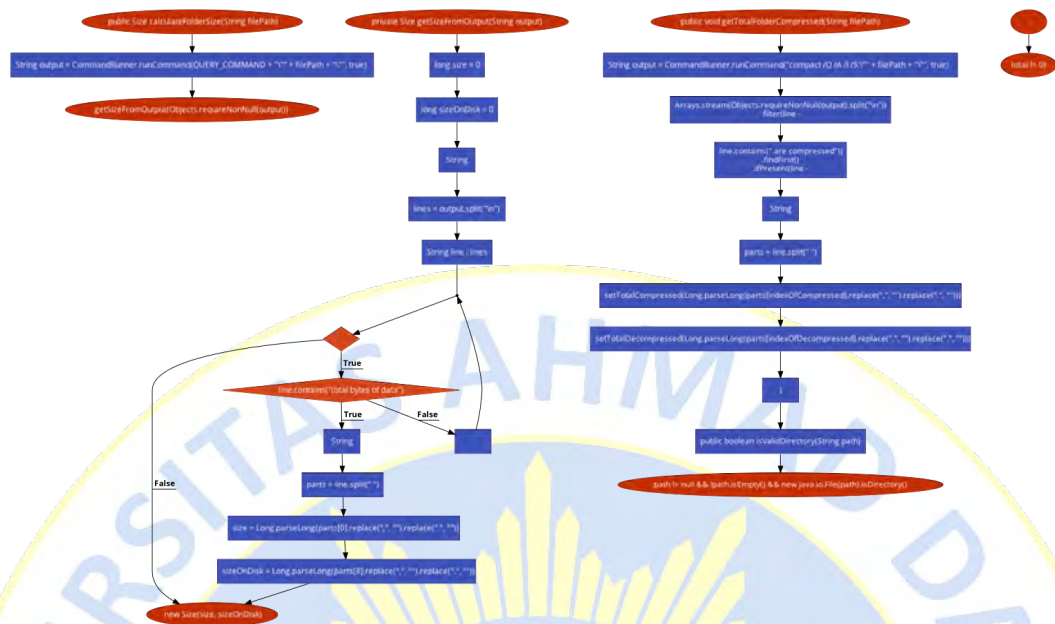
Metode privat `formatSize` digunakan untuk memformat ukuran file berdasarkan satuan yang sesuai. Jika ukuran file kurang dari 1 KB, ukuran tersebut dikembalikan dalam bentuk bytes. Jika ukuran file berada di antara 1 KB dan 1 MB, ukuran tersebut diformat sebagai kilobytes dengan dua angka desimal. Jika ukuran file berada di antara 1 MB dan 1 GB, ukuran tersebut diformat sebagai megabytes dengan dua angka desimal. Jika ukuran file lebih dari 1 GB, ukuran tersebut diformat sebagai gigabytes dengan dua angka desimal.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Misalkan kita memiliki objek `Size` dengan ukuran file sebesar 2048000 bytes (2 MB) dan ukuran file di disk sebesar 2097152 bytes (2 MB). Ketika memanggil metode `getSizeFormatted`, metode `formatSize` akan mengembalikan "2 MB" sebagai hasil. Begitu juga ketika memanggil `getSizeOnDiskFormatted`, hasilnya juga akan "2 MB".

Dengan demikian, kelas `Size` membantu dalam memformat dan menampilkan ukuran file dengan jelas dan sesuai dengan satuan yang mudah dipahami.





Gambar 10 Flowchart class File.java

Flowchart di atas menggambarkan kelas File dalam aplikasi CompactJV, yang bertanggung jawab untuk mengelola operasi terkait file, kompresi, dan informasi sistem.

Pertama-tama, terdapat deklarasi konstanta CMD, QUERY_COMMAND, dan inisialisasi beberapa atribut kelas, seperti filePath, totalCompressed, totalDecompressed, serta objek Compressor yang digunakan untuk operasi kompresi dan dekompresi.

Kemudian, terdapat metode compress dan decompress yang memanfaatkan objek Compressor untuk melakukan kompresi dan dekompresi berdasarkan algoritma yang dipilih.

Selanjutnya, ada dua metode utilitas statis getCPUUsage dan getMemoryUsage yang menggunakan CommandRunner untuk menjalankan perintah WMI dan mendapatkan informasi penggunaan CPU dan memori.

Metode getTotalFilesInFolder menggunakan Files.walk untuk menghitung total file dalam sebuah folder.

Metode isCompressed menggunakan CommandRunner untuk menjalankan perintah compact dan memeriksa apakah file tertentu sudah terkompresi.

Metode calculateFolderSize dan getSizeFromOutput digunakan untuk menghitung ukuran folder dan mengambil hasil dari perintah compact.

Metode getTotalFolderCompressed juga menggunakan CommandRunner untuk mendapatkan informasi jumlah file yang terkompresi dalam sebuah folder.

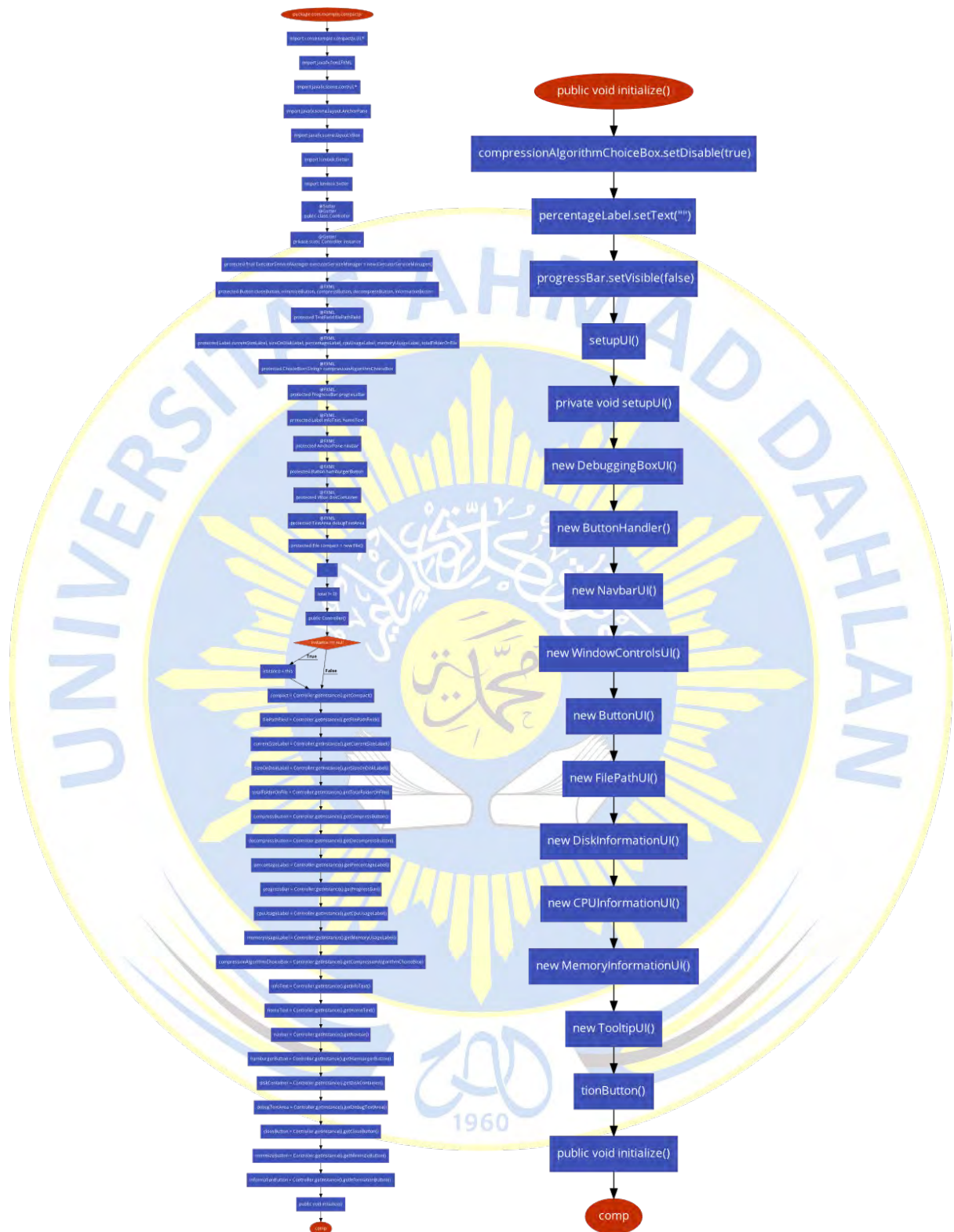
Metode isValidDirectory memeriksa apakah suatu path adalah direktori yang valid.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Misalkan kita memiliki objek File dengan path file yang valid. Ketika memanggil `compress`, metode `compressor.compress` akan dijalankan sesuai dengan algoritma yang dipilih. Begitu juga dengan metode `decompress`, yang akan menjalankan `compressor.decompress`.

Ketika memanggil `getTotalFilesInFolder`, metode tersebut akan menghitung total file dalam suatu folder menggunakan `Files.walk`. Ketika memanggil `isCompressed`, metode tersebut akan menjalankan perintah `compact` dan memeriksa apakah file sudah terkompresi. Ketika memanggil `calculateFolderSize` atau `getTotalFolderCompressed`, metode tersebut akan menjalankan perintah `compact` dan mengambil informasi ukuran folder atau jumlah file terkompresi. Seluruh operasi ini memanfaatkan `CommandRunner` untuk menjalankan perintah dari command prompt secara dinamis.





Gambar 11 Flowchart class Controller.java

Flowchart di atas menggambarkan kelas ``Controller`` dalam aplikasi CompactJV, yang bertanggung jawab untuk mengelola logika bisnis, pengaturan UI, dan interaksi antara komponen-komponen GUI.

Pertama-tama, terdapat inisialisasi atribut-atribut yang merepresentasikan elemen-elemen UI, seperti tombol-tombol, label, choice box, progress bar, dan sebagainya. Selain itu, terdapat inisialisasi objek ``File`` yang digunakan untuk operasi terkait file.

Konstruktor ``Controller`` digunakan untuk mengatur instance dan menginisialisasi objek ``File`` serta beberapa elemen UI agar dapat diakses secara statis.

Metode ``initialize`` dipanggil saat aplikasi dimulai. Metode ini men-disable ``compressionAlgorithmChoiceBox``, mengatur teks pada ``percentageLabel``, dan menyembunyikan ``progressBar``. Selanjutnya, metode ``setupUI`` dipanggil untuk melakukan inisialisasi UI menggunakan berbagai kelas pengatur UI yang terpisah.

Di dalam metode ``setupUI``, terdapat inisialisasi berbagai kelas UI seperti ``DebuggingBoxUI``, ``ButtonHandler``, ``NavbarUI``, ``WindowControlsUI``, ``ButtonUI``, ``FilePathUI``, ``DiskInformationUI``, ``CPUInformationUI``, ``MemoryInformationUI``, dan ``TooltipUI``. Setiap kelas pengatur UI bertanggung jawab untuk mengatur tampilan dan interaksi UI sesuai dengan fungsinya masing-masing.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Ketika aplikasi dimulai, metode ``initialize`` dijalankan untuk mengatur beberapa elemen UI. Selanjutnya, metode ``setupUI`` akan memanggil berbagai kelas pengatur UI untuk melakukan inisialisasi dan mengatur tampilan UI.

Misalnya, ``DebuggingBoxUI`` bertanggung jawab untuk mengatur kotak teks debug, ``ButtonHandler`` untuk mengatur handler tombol, ``NavbarUI`` untuk mengatur tampilan navbar, dan seterusnya.

Ketika pengguna berinteraksi dengan UI, seperti menekan tombol kompresi (``compressButton``), metode yang sesuai akan dijalankan untuk menangani logika bisnis, termasuk operasi kompresi yang diwakili oleh pemanggilan metode pada objek ``File``.

Seluruh alur program dan interaksi UI diatur dan dihubungkan oleh kelas ``Controller``, yang bertindak sebagai pengendali utama aplikasi CompactJV.

Metode `setButtonProperties` menerima dua parameter: tombol yang akan diatur propertinya (`button`) dan handler aksi (`eventHandler`). Handler aksi ini digunakan untuk menangani peristiwa klik pada tombol.

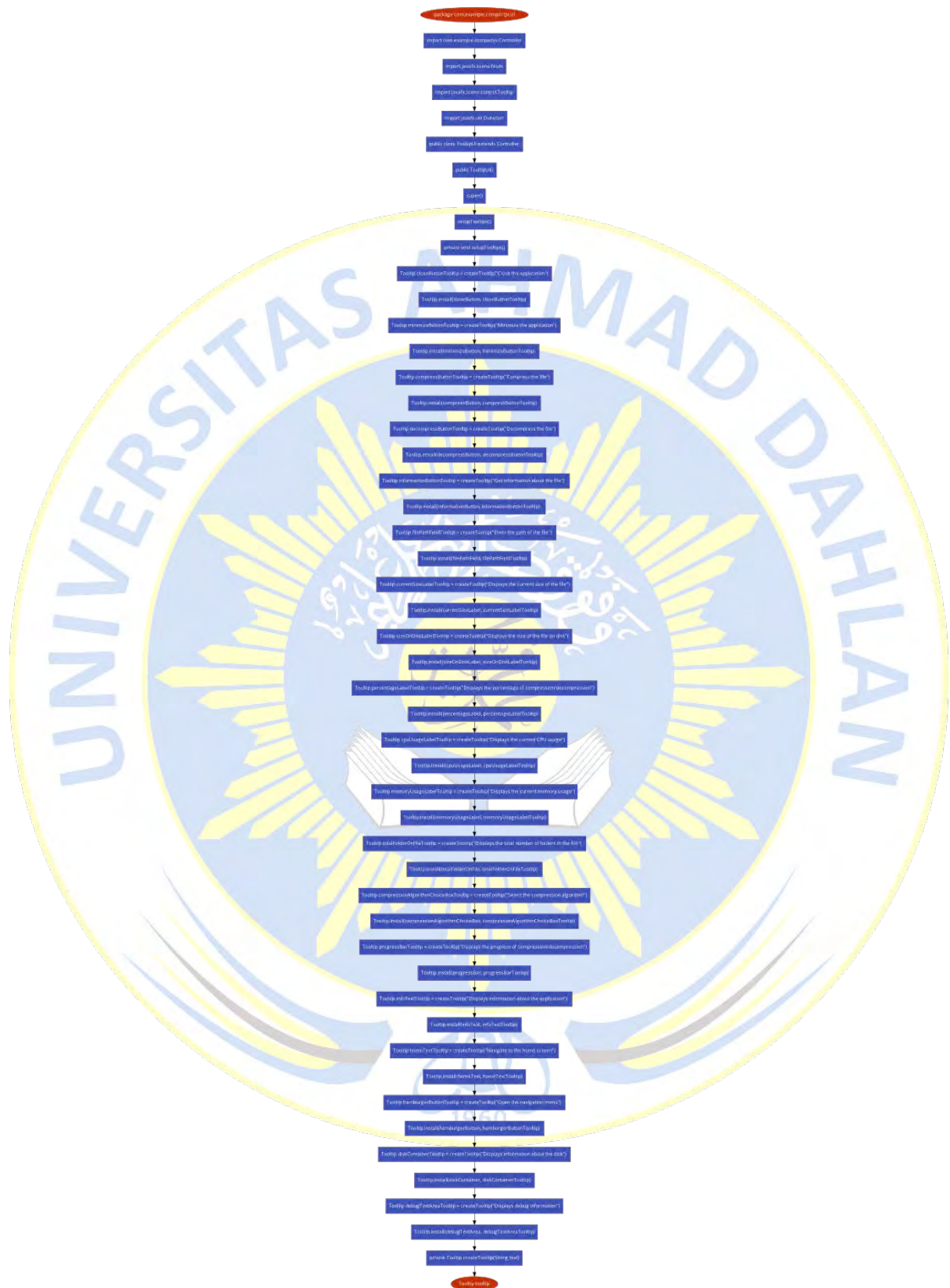
Pada tombol, diterapkan efek animasi dengan menggunakan kelas `ScaleTransition` dari JavaFX. Efek ini memberikan perubahan skala (scale) pada tombol, memberikan kesan visual saat tombol dihover atau diklik. Saat mouse masuk ke area tombol, efek animasi akan memutar ulang dari awal (`playFromStart`). Saat mouse keluar dari area tombol, tombol akan kembali ke skala normal.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Ketika aplikasi CompactJV dijalankan, kelas `WindowControlsUI` akan diinisialisasi, dan metodenya akan dipanggil. Tombol close dan minimize akan diatur propertinya, dan efek animasi akan diterapkan.

Misalnya, jika pengguna mengarahkan mouse ke tombol close, efek animasi akan dimainkan sehingga tombol akan membesar dan menyusut kembali dengan cepat. Jika pengguna mengarahkan mouse keluar dari tombol close, tombol akan kembali ke ukuran normal.

Jika pengguna mengklik tombol close, handler aksi akan dipanggil, dan aplikasi akan keluar (`System.exit(0)`). Jika pengguna mengklik tombol minimize, handler aksi akan dipanggil, dan jendela aplikasi akan di-minimize (`((Stage) ((Node) event.getSource()).getScene().getWindow()).setIconified(true)`).



Gambar 13 Flowchart class TooltipUI

Flowchart di atas menggambarkan kelas `TooltipUT` yang merupakan bagian dari antarmuka pengguna (UI) dalam aplikasi CompactJV. Kelas ini bertanggung jawab untuk mengatur tooltip pada berbagai elemen antarmuka pengguna, seperti tombol, label, dan choice box.

Ketika objek `TooltipUT` diinisialisasi, konstruktor akan memanggil metode `setupTooltips` untuk mengatur tooltips pada berbagai elemen antarmuka pengguna.

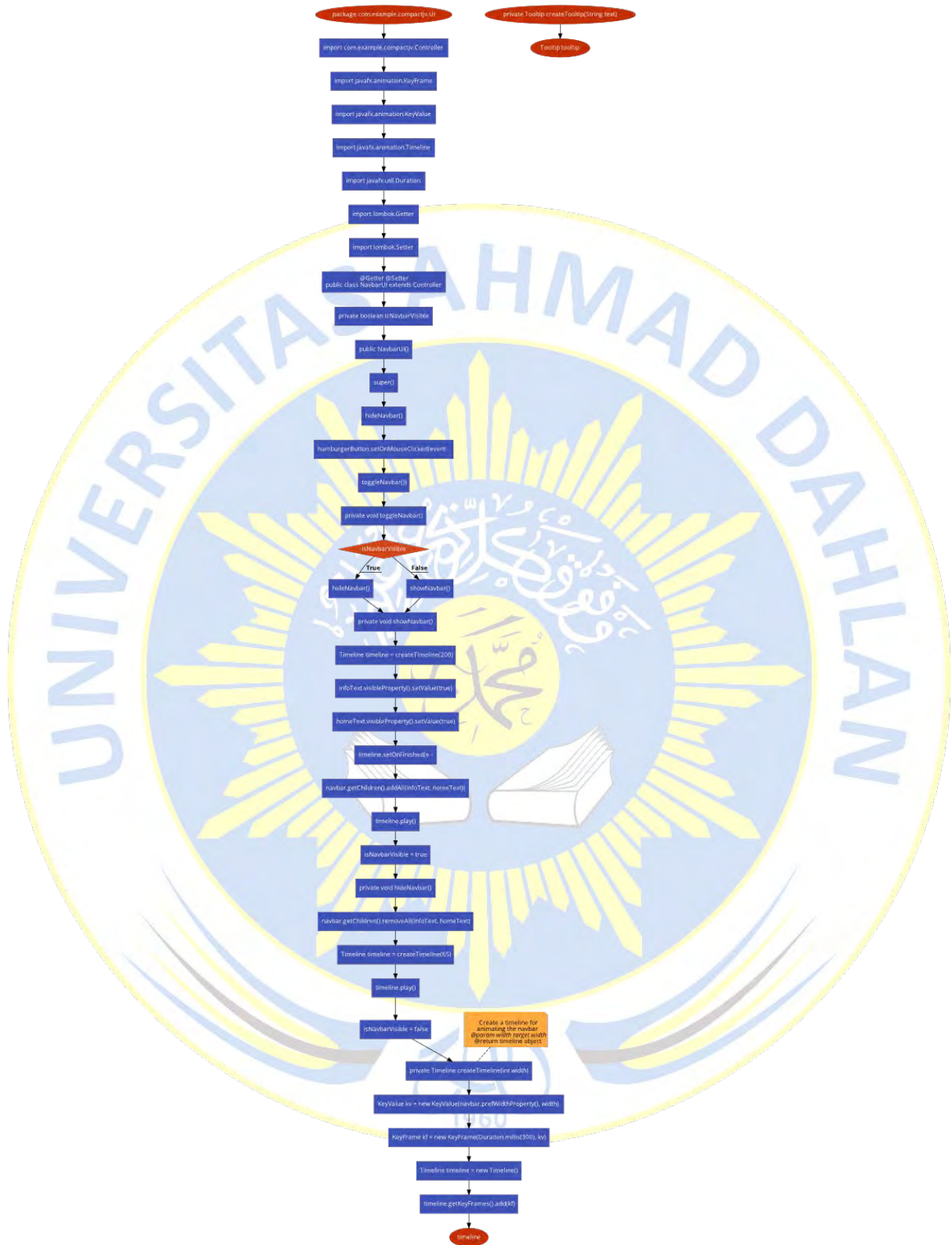
Metode `setupTooltips` akan membuat tooltips untuk setiap elemen antarmuka pengguna, seperti tombol close, minimize, compress, decompress, dan sebagainya. Setiap tooltip akan diatur menggunakan metode `createTooltip`, yang menerima teks sebagai parameter.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Ketika aplikasi CompactJV dijalankan, kelas `TooltipUT` akan diinisialisasi, dan metodenya akan dipanggil. Metode `setupTooltips` akan membuat dan mengatur tooltips untuk setiap elemen antarmuka pengguna yang dijelaskan dalam kodingan.

Misalnya, ketika pengguna mengarahkan mouse ke tombol close, tooltip akan muncul dengan teks "Close the application" setelah penundaan selama 0.5 detik. Tooltip ini akan mengikuti posisi mouse yang bergerak, memberikan informasi yang berguna kepada pengguna.

Proses ini diulang untuk setiap elemen antarmuka pengguna yang membutuhkan tooltip. Hal ini meningkatkan pengalaman pengguna dengan memberikan informasi yang jelas dan berguna saat mereka berinteraksi dengan berbagai fitur aplikasi.



Gambar 14 Flowchart class NavbarUI.java

Flowchart di atas menggambarkan kelas NavbarUI dalam aplikasi CompactJV. Kelas ini bertanggung jawab untuk menangani tampilan dan animasi dari navbar pada antarmuka pengguna.

Pertama-tama, saat objek NavbarUI diinisialisasi, konstruktor akan memanggil metode hideNavbar untuk menyembunyikan navbar secara default. Selain itu, tombol hamburger pada antarmuka pengguna akan memiliki event handler yang memanggil metode toggleNavbar ketika diklik.

Metode toggleNavbar akan memeriksa apakah navbar sedang terlihat atau tidak. Jika terlihat, metode hideNavbar akan dipanggil untuk menyembunyikannya. Jika tidak terlihat, metode showNavbar akan dipanggil untuk menampilkannya.

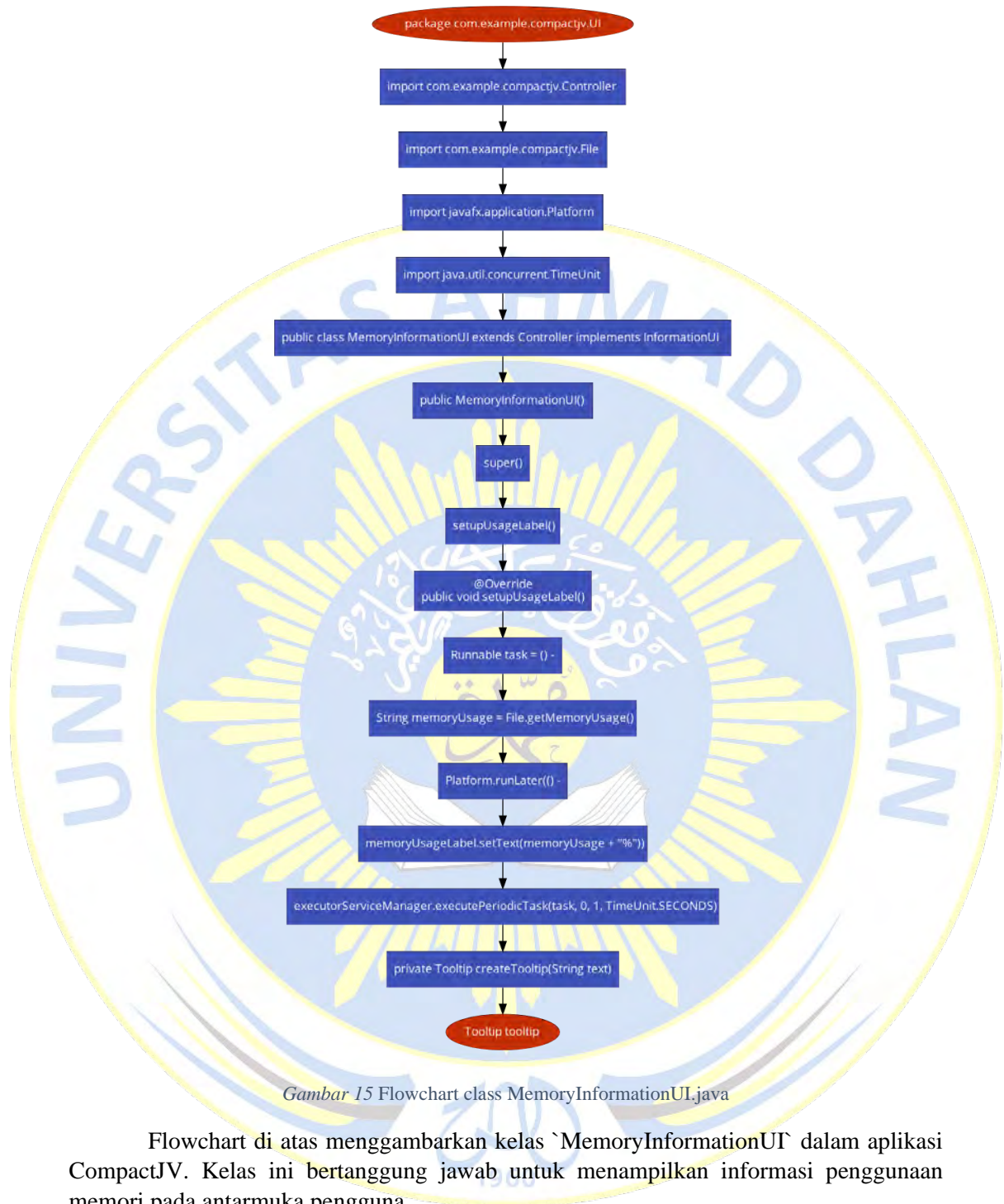
Metode showNavbar membuat objek Timeline yang berisi animasi untuk menampilkan navbar dengan lebar yang sesuai. Selain itu, elemen-elemen seperti infoText dan homeText diatur menjadi terlihat, dan mereka ditambahkan ke dalam container navbar. Setelah animasi selesai, properti isVisible diatur menjadi true.

Metode hideNavbar menghapus elemen-elemen dari container navbar dan membuat objek Timeline untuk animasi menyembunyikan navbar dengan lebar yang sesuai. Setelah animasi selesai, properti isVisible diatur menjadi false.

Terakhir, metode createTimeline digunakan untuk membuat objek Timeline dengan durasi animasi 300 milidetik, berdasarkan target lebar yang diberikan sebagai parameter.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Ketika aplikasi CompactJV dijalankan, objek NavbarUI akan diinisialisasi. Saat pengguna mengklik tombol hamburger, metode toggleNavbar akan memeriksa status navbar. Jika terlihat, maka metode hideNavbar akan dipanggil untuk menyembunyikannya. Jika tidak terlihat, maka metode showNavbar akan dipanggil untuk menampilkannya. Animasi akan memberikan efek transisi lembut saat menampilkan atau menyembunyikan navbar, meningkatkan pengalaman pengguna.



Flowchart di atas menggambarkan kelas `MemoryInformationUI` dalam aplikasi CompactJV. Kelas ini bertanggung jawab untuk menampilkan informasi penggunaan memori pada antarmuka pengguna.

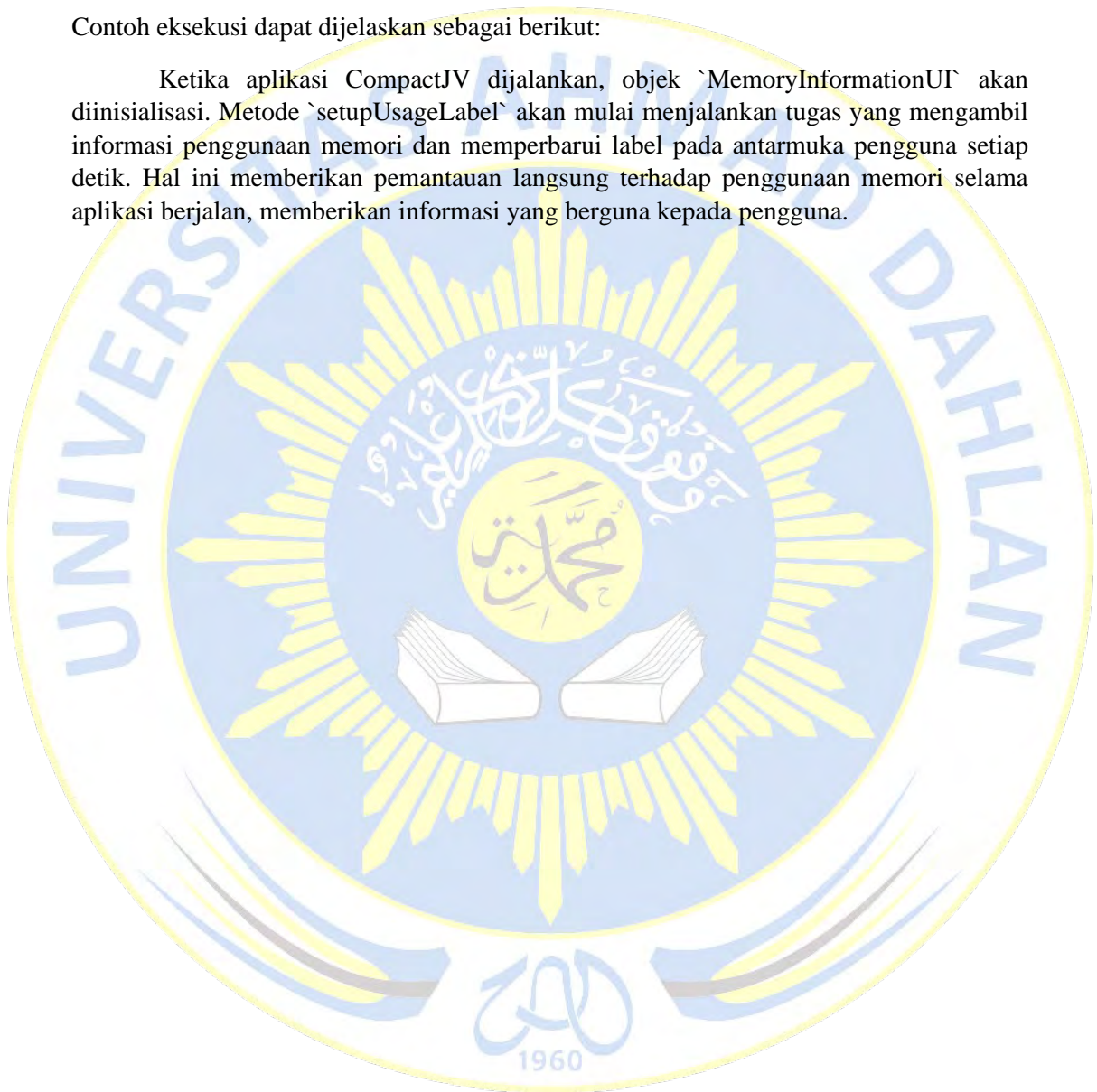
Pertama-tama, saat objek `MemoryInformationUI` diinisialisasi, konstruktor akan memanggil metode `setupUsageLabel`. Metode ini mengimplementasikan antarmuka `InformationUI` dan menggunakan `Runnable task` untuk menjalankan suatu tugas secara berkala.

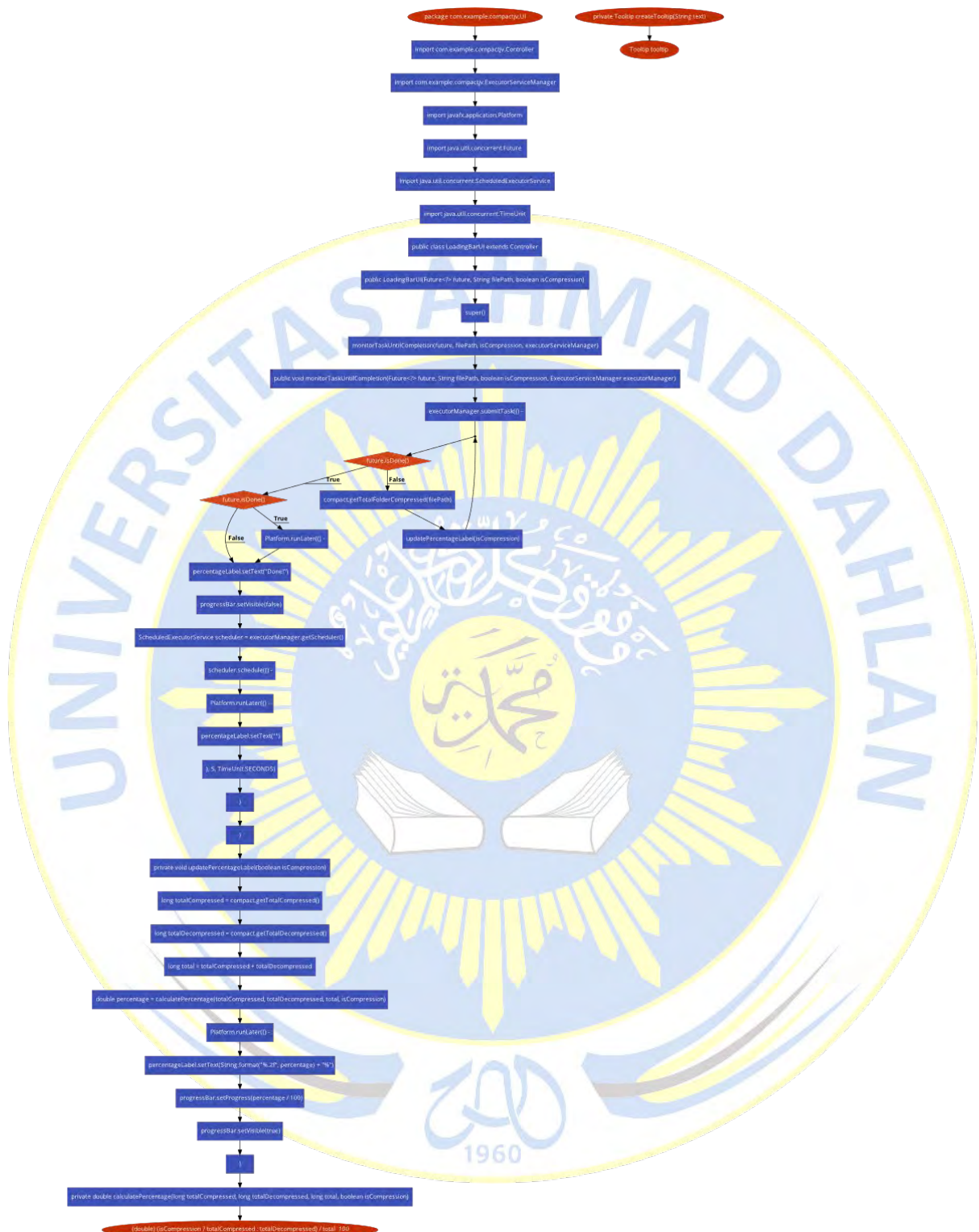
Tugas tersebut memanggil metode statis `getMemoryUsage` dari kelas `File` untuk mendapatkan informasi penggunaan memori. Informasi tersebut kemudian ditampilkan pada label `memoryUsageLabel` di antarmuka pengguna.

Metode `executePeriodicTask` dari `executorServiceManager` digunakan untuk menjalankan tugas tersebut secara berkala dengan interval 1 detik. Ini memastikan bahwa informasi penggunaan memori diperbarui secara terus-menerus.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Ketika aplikasi CompactJV dijalankan, objek `MemoryInformationUI` akan diinisialisasi. Metode `setupUsageLabel` akan mulai menjalankan tugas yang mengambil informasi penggunaan memori dan memperbarui label pada antarmuka pengguna setiap detik. Hal ini memberikan pemantauan langsung terhadap penggunaan memori selama aplikasi berjalan, memberikan informasi yang berguna kepada pengguna.





Gambar 16 Flowchart class LoadingBarUI.java

Flowchart di atas menggambarkan kelas `LoadingBarUI` dalam aplikasi CompactJV. Kelas ini bertanggung jawab untuk memantau dan menampilkan status kemajuan tugas kompresi atau dekompresi melalui antarmuka pengguna.

Ketika objek `LoadingBarUI` diinisialisasi dengan memberikan `Future<?> future`, `String filePath`, dan `boolean isCompression` sebagai parameter, konstruktor akan memanggil metode `monitorTaskUntilCompletion` untuk memantau tugas yang sedang berjalan.

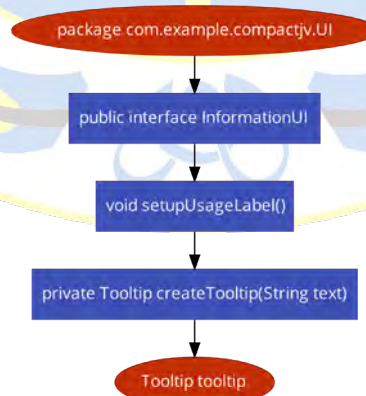
Metode `monitorTaskUntilCompletion` akan menjalankan tugas secara berkala untuk memperbarui label persentase dan batang progres pada antarmuka pengguna selama tugas kompresi atau dekompresi sedang berlangsung. Hal ini dicapai dengan menggunakan `ScheduledExecutorService` dari `executorServiceManager` untuk menjalankan tugas dalam thread terpisah.

Selama tugas belum selesai (`!future.isDone()`), metode ini akan memperbarui label persentase dan batang progres dengan menggunakan metode `updatePercentageLabel`. Metode ini menghitung persentase kemajuan berdasarkan jumlah file yang sudah dikompresi atau didekompresi dibandingkan dengan total file yang akan diproses.

Jika tugas telah selesai (`future.isDone()`), antarmuka pengguna akan diperbarui untuk menampilkan pesan "Done!" pada label persentase. Selain itu, batang progres akan disembunyikan, dan setelah 5 detik, label persentase akan dikosongkan kembali.

Contoh eksekusi dapat dijelaskan sebagai berikut:

Misalkan Anda melakukan kompresi file dengan objek `LoadingBarUI` dan memberikan tugas kompresi yang diwakili oleh `future`. Selama tugas kompresi berlangsung, antarmuka pengguna akan menampilkan persentase kemajuan dan batang progres yang terus diperbarui secara real-time. Begitu tugas selesai, antarmuka pengguna akan menampilkan pesan "Done!" dan secara otomatis menghilangkan batang progres setelah beberapa detik. Ini memberikan pengalaman pengguna yang jelas dan informatif selama proses kompresi atau dekompresi berlangsung.



Gambar 17 Flowchart class InformationUI.java

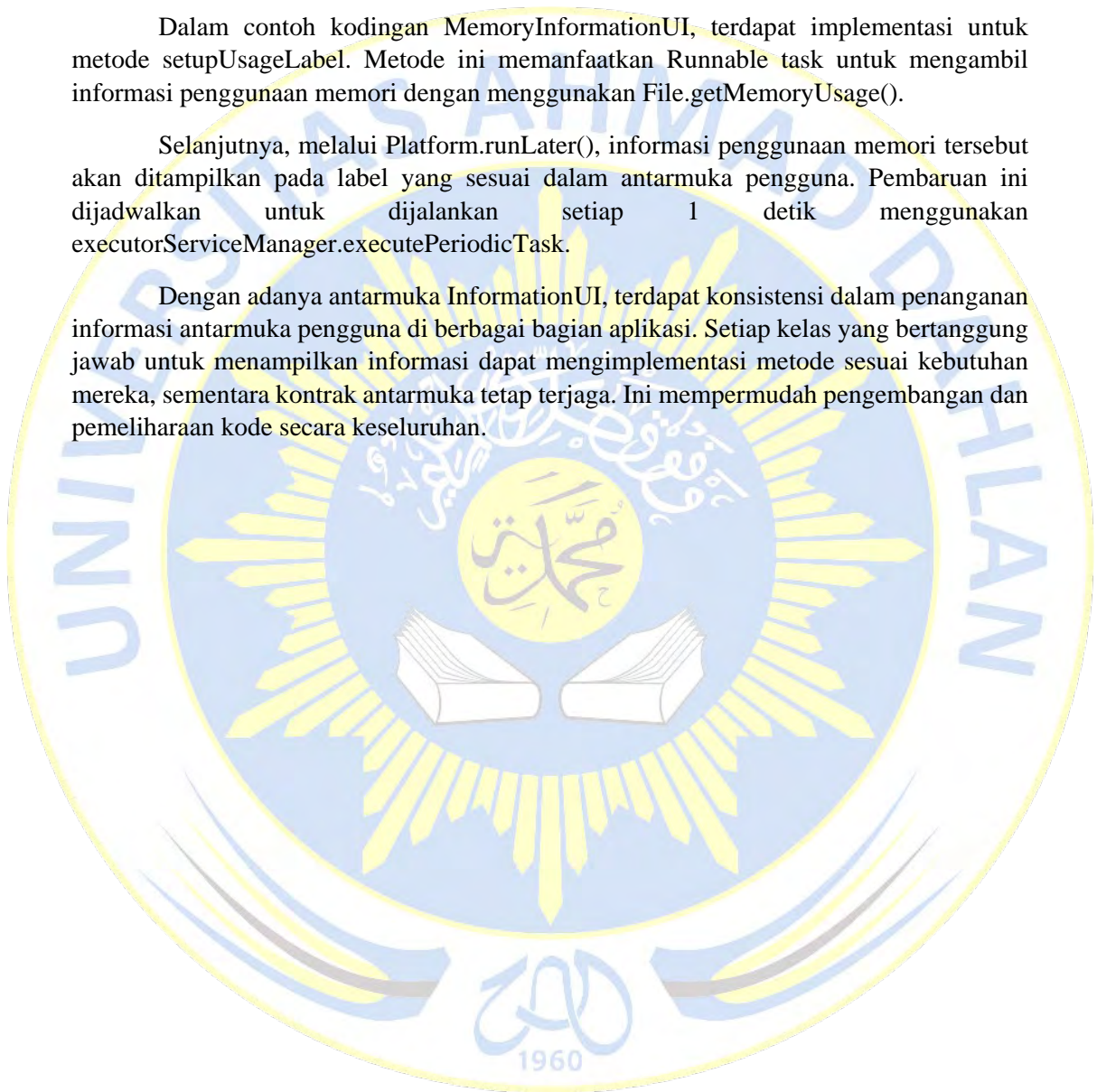
Flowchart di atas mencerminkan antarmuka InformationUI dalam aplikasi CompactJV. Antarmuka ini memiliki satu metode, yaitu `setupUsageLabel`, yang nantinya akan diimplementasikan oleh kelas yang mengimplementasi antarmuka ini.

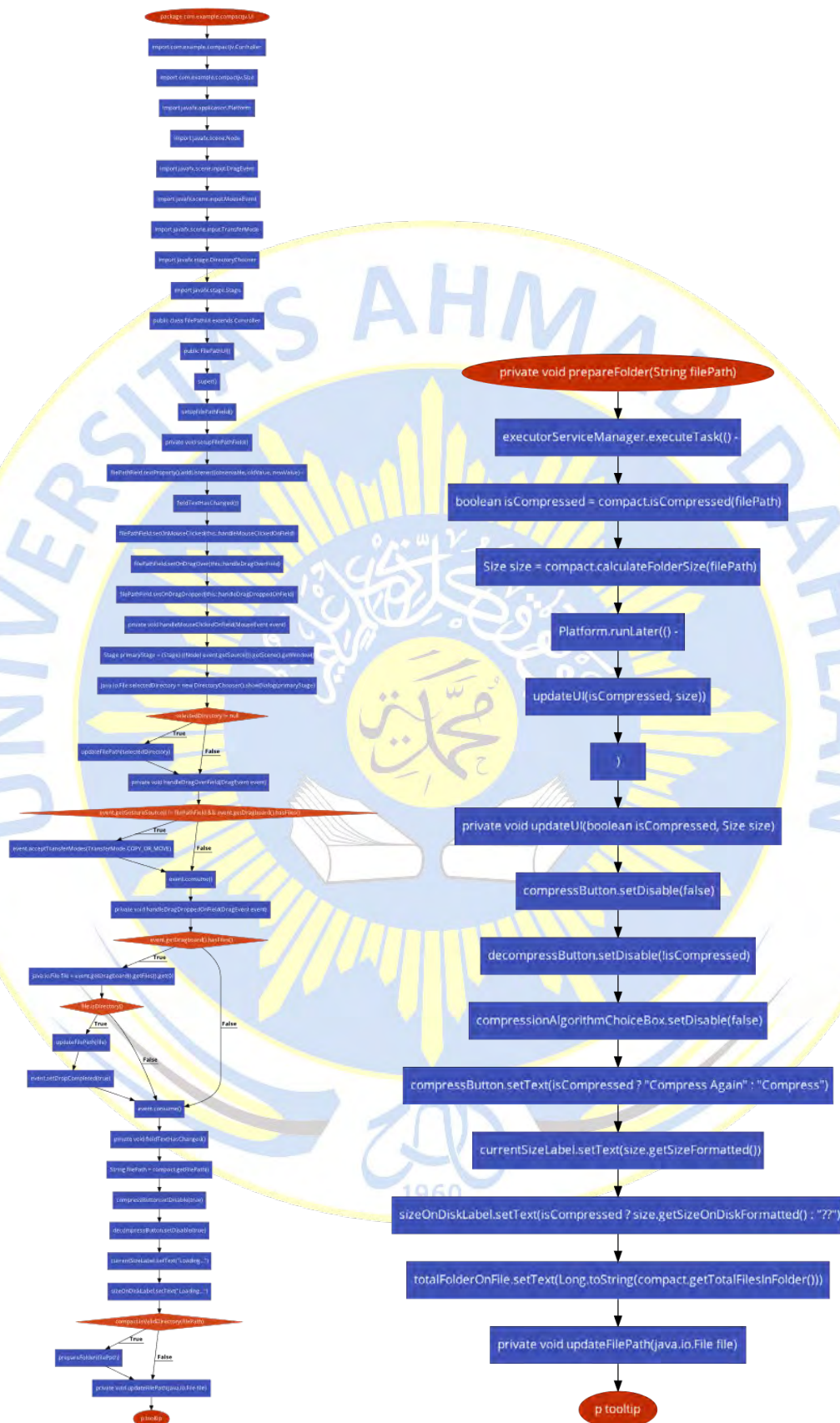
Ketika kelas, seperti contohnya `MemoryInformationUI` yang telah diberikan sebelumnya, mengimplementasi antarmuka `InformationUI`, kelas tersebut wajib menyediakan implementasi untuk metode `setupUsageLabel`.

Dalam contoh kodingan `MemoryInformationUI`, terdapat implementasi untuk metode `setupUsageLabel`. Metode ini memanfaatkan `Runnable` task untuk mengambil informasi penggunaan memori dengan menggunakan `File.getMemoryUsage()`.

Selanjutnya, melalui `Platform.runLater()`, informasi penggunaan memori tersebut akan ditampilkan pada label yang sesuai dalam antarmuka pengguna. Pembaruan ini dijadwalkan untuk dijalankan setiap 1 detik menggunakan `executorServiceManager.executePeriodicTask`.

Dengan adanya antarmuka `InformationUI`, terdapat konsistensi dalam penanganan informasi antarmuka pengguna di berbagai bagian aplikasi. Setiap kelas yang bertanggung jawab untuk menampilkan informasi dapat mengimplementasi metode sesuai kebutuhan mereka, sementara kontrak antarmuka tetap terjaga. Ini mempermudah pengembangan dan pemeliharaan kode secara keseluruhan.





Gambar 18 Flowchart class FilePath.java

Flowchart untuk kelas `FilePathUI` dalam aplikasi `CompactJV` dimulai dengan konstruktor yang dipanggil saat objek dari kelas tersebut dibuat. Konstruktor tersebut kemudian memanggil konstruktor dari kelas induk (`Controller`) dan metode `setupFilePathField()` untuk menyiapkan field path file.

Dalam metode `setupFilePathField()`, ditambahkan listener pada properti teks dari field path file (`filePathField`) agar dapat merespons perubahan nilai field dengan memanggil metode `fieldTextHasChanged()`. Selain itu, ditetapkan event handler untuk klik mouse (`handleMouseClickedOnField`), drag over (`handleDragOverField`), dan drop saat drag (`handleDragDroppedOnField`).

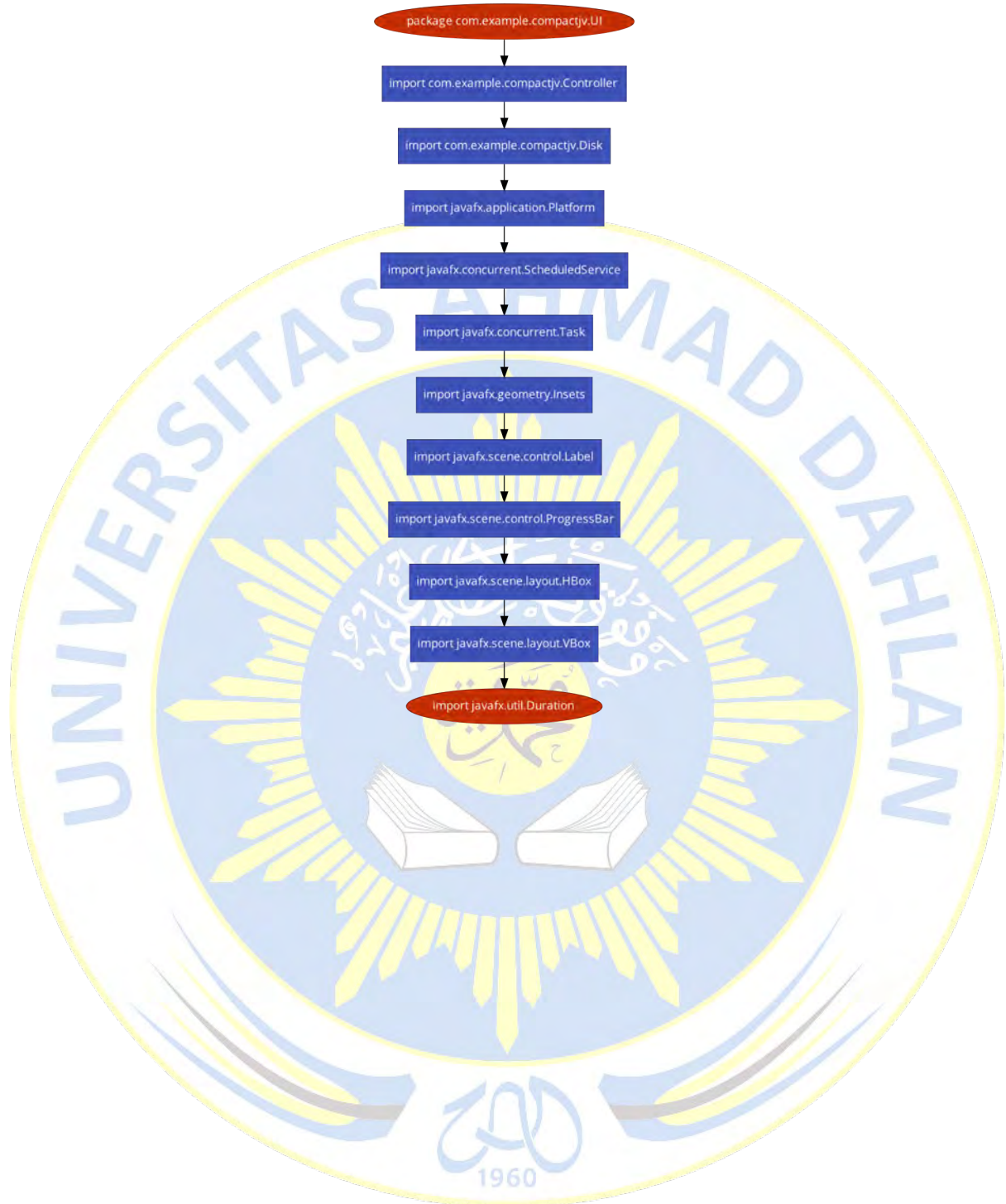
Event handler untuk klik mouse (`handleMouseClickedOnField`) memperoleh stage utama dari sumber event, menampilkan dialog pemilihan direktori, dan jika direktori dipilih, memanggil metode `updateFilePath()` untuk memperbarui path file. Event handler untuk drag over (`handleDragOverField`) memeriksa apakah sumber drag bukan field path file dan terdapat file yang di-drag, jika ya, menerima mode transfer `COPY` atau `MOVE`. Event handler untuk drop (`handleDragDroppedOnField`) memeriksa apakah terdapat file yang di-drag, dan jika ya, memperoleh file pertama dari daftar file yang di-drag. Jika file adalah direktori, metode `updateFilePath()` dipanggil untuk memperbarui path file.

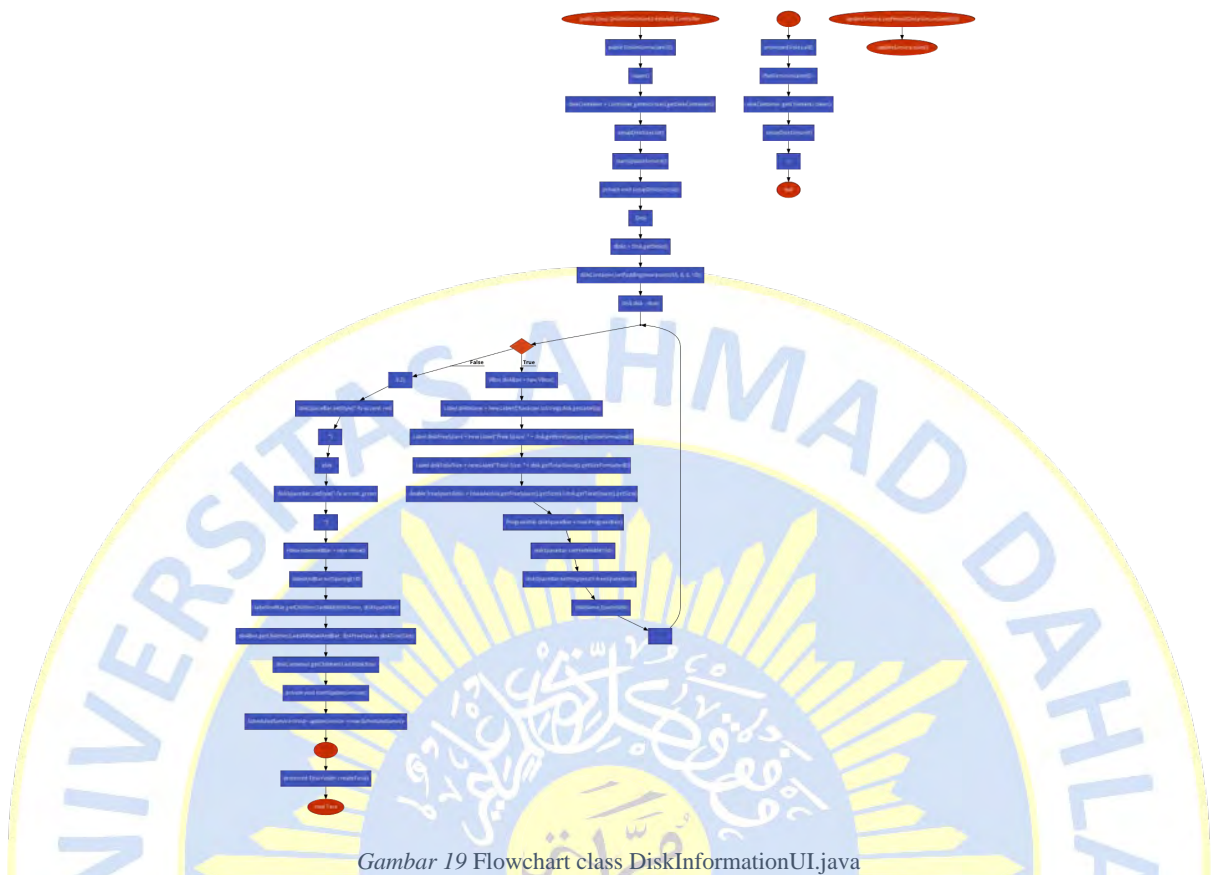
Metode `fieldTextHasChanged()` dipanggil setiap kali nilai teks field path file berubah. Ini digunakan untuk mendapatkan path file dari objek `compact`, menonaktifkan tombol kompres dan dekompres, serta menetapkan label ukuran saat ini dan ukuran di disk menjadi "Loading...". Jika direktori valid, metode `prepareFolder()` dipanggil.

Metode `updateFilePath(java.io.File file)` digunakan untuk memperbarui path file di objek `compact` dan mengatur teks field path file. Metode `prepareFolder(String filePath)` menjalankan tugas pada `executor service` untuk menentukan apakah direktori tersebut sudah dikompres, menghitung ukuran folder, dan menjalankan tugas di thread UI untuk memperbarui UI.

Terakhir, metode `updateUI(boolean isCompressed, Size size)` digunakan untuk mengatur UI dengan mengaktifkan atau menonaktifkan tombol sesuai kondisi, mengatur teks pada tombol kompres, label ukuran saat ini, ukuran di disk, dan jumlah total file dalam folder.

Alur program ini dirancang untuk memberikan respons yang cepat dan informatif terhadap tindakan pengguna terkait path file, baik melalui klik mouse, drag-and-drop, maupun perubahan nilai teks. `Executor service` digunakan untuk menjalankan tugas di latar belakang agar antarmuka pengguna tetap responsif.





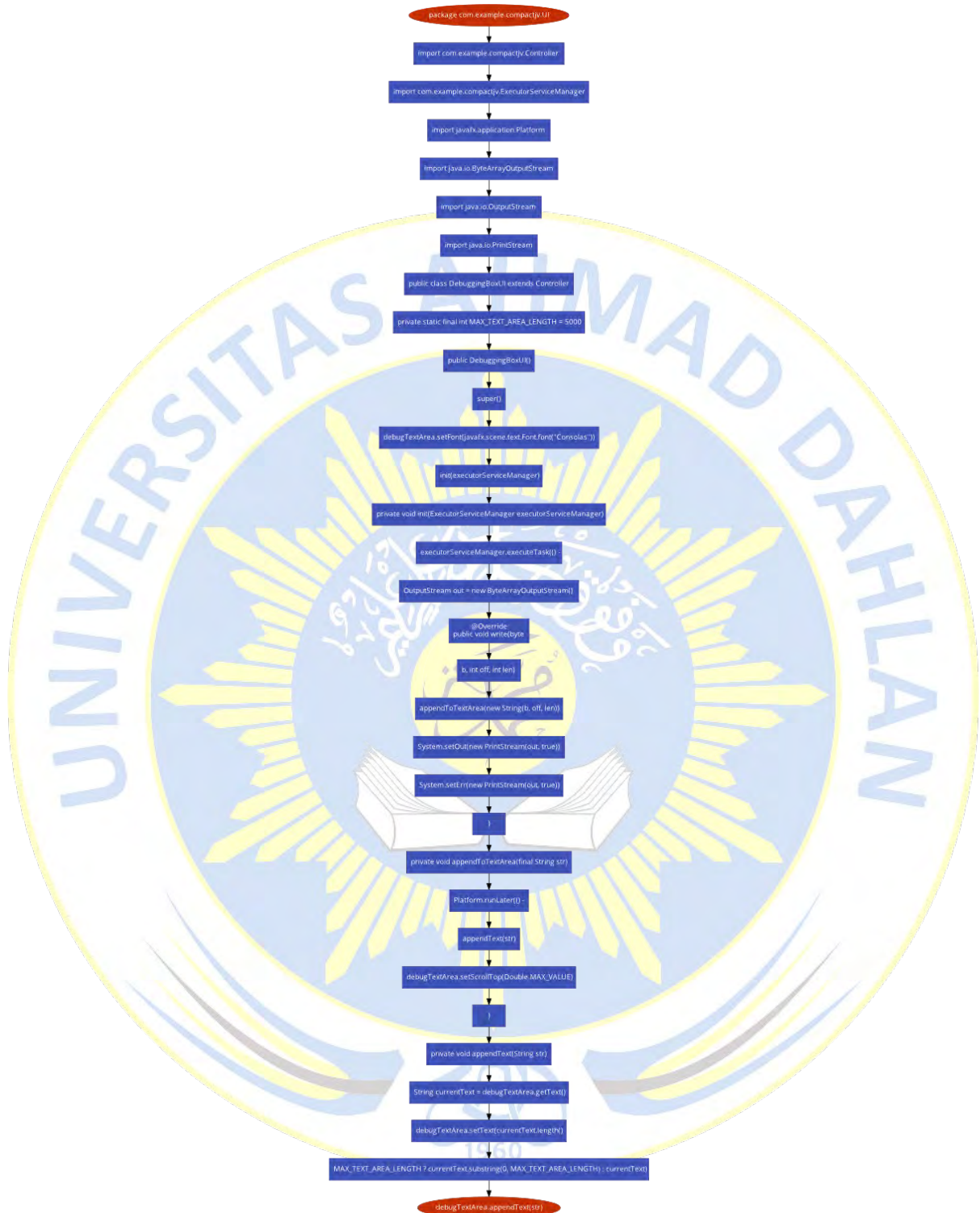
Gambar 19 Flowchart class DiskInformationUI.java

Flowchart untuk kelas `DiskInformationUI` dalam aplikasi CompactJV dimulai dengan konstruktor yang dipanggil saat objek dari kelas tersebut dibuat. Konstruktor tersebut memanggil konstruktor dari kelas induk (`Controller`), mengakses objek `diskContainer` dari kelas `Controller`, dan memanggil metode `setupDiskSizeList()` dan `startUpdateService()`.

Dalam metode `setupDiskSizeList()`, objek `Disk` diambil untuk mendapatkan informasi tentang semua disk yang tersedia. Selanjutnya, untuk setiap disk, sebuah `VBox` baru dibuat untuk menampung informasi disk. Label-label seperti nama disk, ruang kosong, dan ukuran total disk ditambahkan ke dalam `VBox`. Sebuah `ProgressBar` juga dibuat untuk menunjukkan rasio ruang kosong terhadap ukuran total disk. Warna dan gaya dari elemen-elemen ini disesuaikan berdasarkan rasio tersebut. Semua `VBox` yang berisi informasi disk ditambahkan ke dalam `diskContainer`.

Metode `startUpdateService()` digunakan untuk membuat dan menjalankan layanan terjadwal (`ScheduledService`) yang memperbarui informasi disk setiap 5 detik. Layanan ini menjalankan tugas di thread latar belakang, dan dengan menggunakan `Platform.runLater()`, memastikan bahwa pembaruan dilakukan di thread UI untuk menghindari konflik thread.

Dengan alur ini, kelas `DiskInformationUI` memberikan informasi disk yang real-time dengan tampilan yang dinamis sesuai dengan perubahan ruang kosong pada setiap disk. Layanan terjadwal memastikan pembaruan terus menerus tanpa membebani UI.



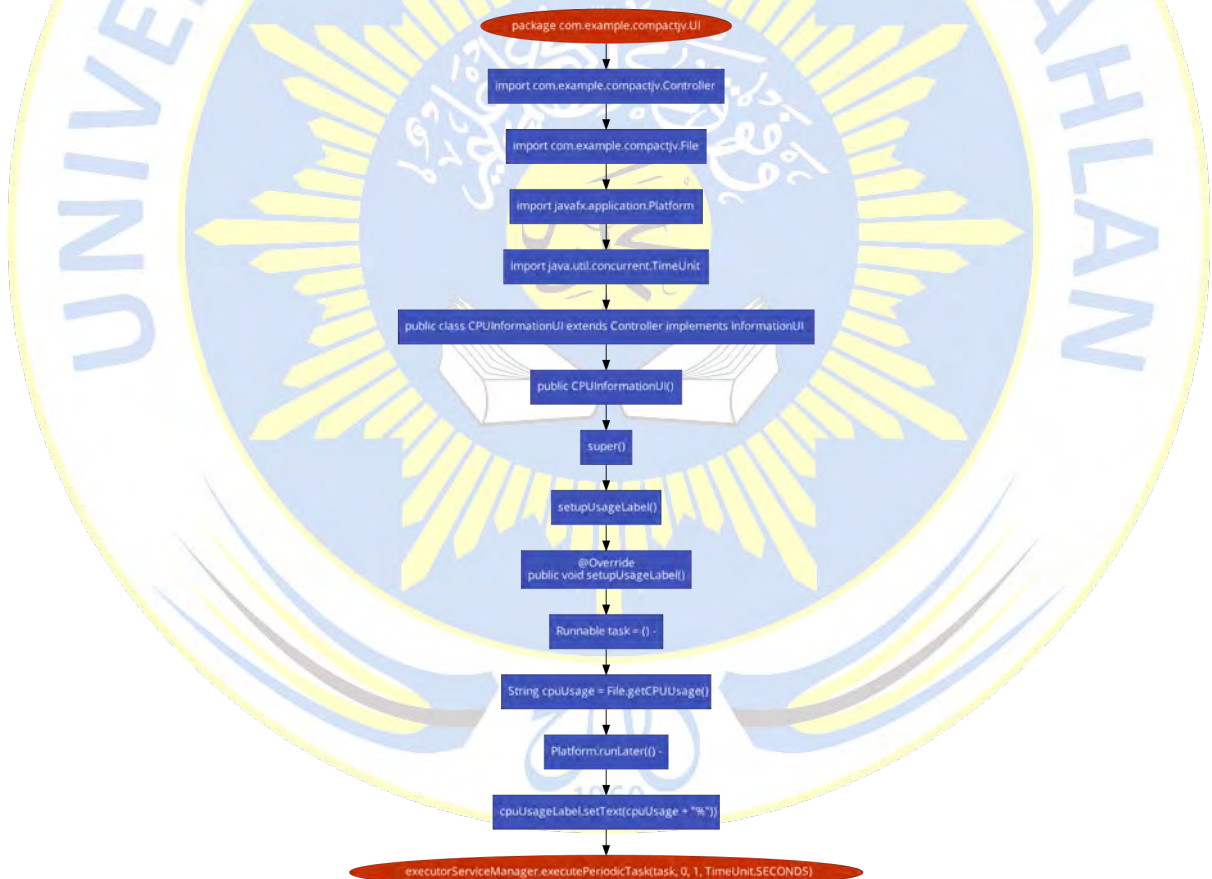
Gambar 20 Flowchart class DebuggingBoxUI.java

Flowchart untuk kelas `DebuggingBoxUI` dalam aplikasi `CompactJV` dimulai dengan konstruktor yang dipanggil saat objek dari kelas tersebut dibuat. Konstruktor tersebut memanggil konstruktor dari kelas induk (`Controller`), mengakses objek `debugTextArea` dari kelas `Controller`, dan memanggil metode `init()`.

Dalam metode `init()`, objek `OutputStream` digunakan untuk menangkap output dari `System.out` dan `System.err`. Metode `appendToTextArea()` dipanggil untuk menambahkan output tersebut ke dalam `debugTextArea`. Penggunaan `Platform.runLater()` memastikan bahwa operasi ini dilakukan di thread UI untuk menghindari konflik thread.

Metode `appendToTextArea()` mengatur teks dalam `debugTextArea` dengan memanggil `appendText()`. Metode `appendText()` memastikan bahwa teks dalam `debugTextArea` tidak melebihi batas `MAX_TEXT_AREA_LENGTH`. Jika melebihi, teks yang lebih lama akan dipotong.

Dengan alur ini, kelas `DebuggingBoxUI` memberikan antarmuka pengguna untuk melihat dan memantau output debugging dari aplikasi `CompactJV`. Pengguna dapat melihat pesan debugging secara real-time dan melakukan scrolling jika diperlukan.



Gambar 21 Flowchart Class `CPUInformationUI.java`

Dalam flowchart kelas `CPUInformationUI` pada aplikasi `CompactJV`, konstruktor pertama kali dipanggil ketika objek dari kelas ini dibuat. Konstruktor tersebut memanggil

konstruktor dari kelas induk (Controller) dan kemudian memanggil metode `setupUsageLabel()`.

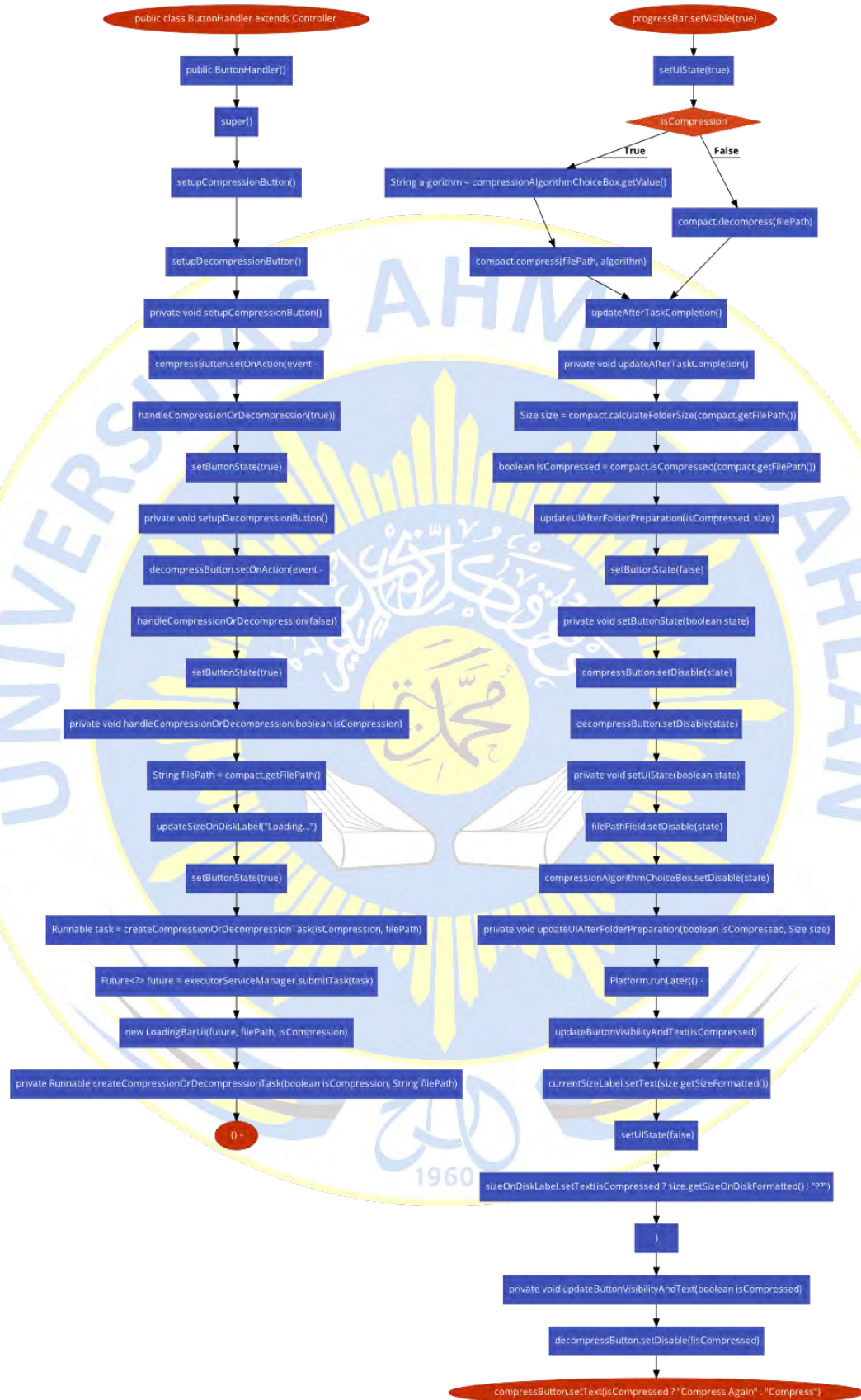
Metode `setupUsageLabel()` mengimplementasikan antarmuka `InformationUI`. Pada langkah pertama, sebuah objek `Runnable` dibuat untuk merepresentasikan tugas yang akan dijalankan. Tugas ini bertanggung jawab untuk mendapatkan informasi penggunaan CPU dan mengupdate label pada antarmuka pengguna (`cpuUsageLabel`) dengan nilai tersebut.

Selanjutnya, tugas tersebut dijadwalkan untuk dijalankan secara periodik menggunakan `executorServiceManager.executePeriodicTask()`. Task ini dijalankan setiap 1 detik (dalam contoh ini) dan menggunakan `File.getCPUUsage()` untuk mendapatkan persentase penggunaan CPU.

Dengan menggunakan `Platform.runLater()`, pembaruan label dilakukan di thread UI, sehingga tidak terjadi konflik thread.

Dengan alur ini, kelas `CPUInformationUI` memberikan antarmuka pengguna yang terus-menerus mengupdate informasi penggunaan CPU secara real-time. Hal ini memberikan pemahaman yang jelas kepada pengguna tentang seberapa intensif penggunaan CPU oleh aplikasi `CompactJV`.





Gambar 22 Flowchart class buttonHandlerUI.java

Dalam flowchart kelas ButtonHandler pada aplikasi CompactJV, konstruktor pertama kali dipanggil saat objek dari kelas ini dibuat. Konstruktor tersebut memanggil konstruktor dari kelas induk (Controller) dan kemudian memanggil metode `setupCompressionButton()` dan `setupDecompressionButton()`.

Metode `setupCompressionButton()` dan `setupDecompressionButton()` bertanggung jawab untuk mengatur aksi yang terjadi saat tombol kompresi dan dekompresi ditekan. Kedua metode ini mendaftarkan handler acara untuk tombol-tombol tersebut dan memanggil metode `handleCompressionOrDecompression()` dengan parameter yang sesuai (true untuk kompresi, false untuk dekompresi).

Metode `handleCompressionOrDecompression(boolean isCompression)` merupakan inti dari kelas ini. Saat tombol kompresi atau dekompresi ditekan, metode ini dijalankan. Pertama, itu mengambil jalur file dari objek CompactJV. Kemudian, label ukuran di disk diupdate dengan teks "Loading...", dan status tombol diatur menjadi nonaktif.

Selanjutnya, metode ini membuat tugas yang akan dijalankan oleh `executorServiceManager` menggunakan `createCompressionOrDecompressionTask()`. Tugas tersebut akan menampilkan batang kemajuan, mengunci antarmuka pengguna, dan kemudian memanggil metode kompresi atau dekompresi pada objek CompactJV sesuai dengan parameter yang diberikan.

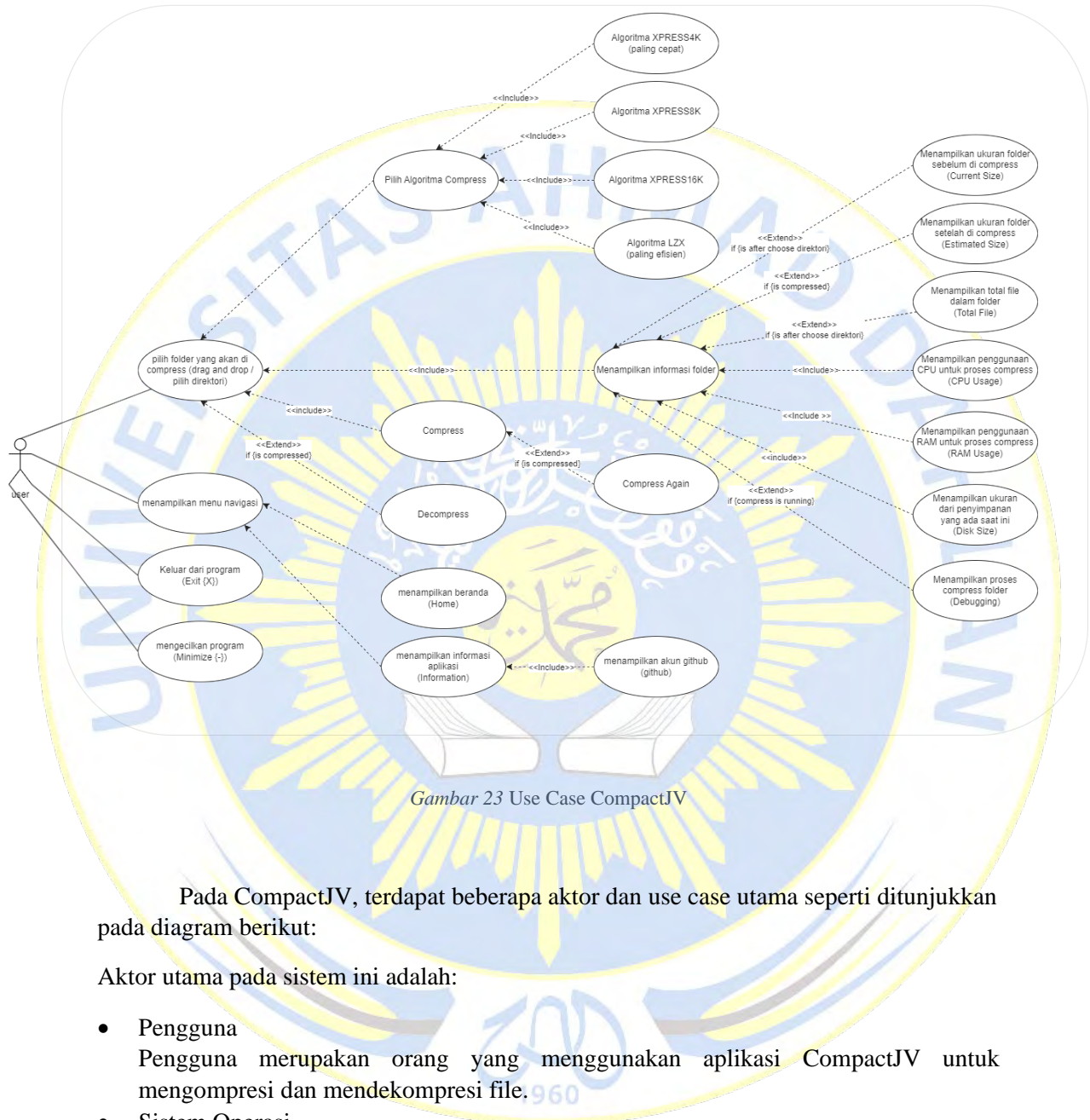
Setelah tugas selesai, metode `updateAfterTaskCompletion()` dipanggil. Ini menghitung ukuran folder yang baru dikompres atau didekompres, mengupdate antarmuka pengguna sesuai dengan hasilnya, dan kemudian mengatur kembali status tombol dan antarmuka pengguna.

Metode-metode bantuan seperti `updateSizeOnDiskLabel()`, `setButtonState()`, `setUIState()`, dan `updateUIAfterFolderPreparation()` digunakan untuk memperbarui elemen UI, mengatur status tombol dan antarmuka pengguna, dan memberikan pembaruan visual setelah tugas selesai.

Dengan alur ini, kelas ButtonHandler memberikan pengguna kemampuan untuk mengompres dan mendekompres file atau folder dengan antarmuka pengguna yang responsif dan menyajikan informasi yang jelas tentang prosesnya.

2.3. Use Case Diagram

compactJV use case



Gambar 23 Use Case CompactJV

Pada CompactJV, terdapat beberapa aktor dan use case utama seperti ditunjukkan pada diagram berikut:

Aktor utama pada sistem ini adalah:

- **Pengguna**
Pengguna merupakan orang yang menggunakan aplikasi CompactJV untuk mengompresi dan mendekompresi file.
- **Sistem Operasi**
Sistem operasi tempat aplikasi CompactJV dijalankan, dalam hal ini Windows. Sistem operasi menyediakan akses ke berbagai fitur seperti file system dan exec() untuk running program.
- **External Tools**
Perangkat lunak eksternal seperti compact.exe yang digunakan oleh CompactJV untuk melakukan kompresi dan dekompresi file secara aktual.

Sedangkan use case utama pada sistem adalah:

- Memilih File atau Folder

Use case ini berinteraksi langsung dengan aktor Pengguna, dimana pengguna ingin menentukan file/folder yang akan diproses CompactJV. Ada dua cara untuk memilih file/folder yaitu pertama dengan menginputkan path folder ke text field pada antarmuka CompactJV. Kedua dengan melakukan drag & drop file/folder ke dalam jendela aplikasi. File atau folder yang dipilih bisa berupa path di mana saja pada komputer, misalnya:

C:\Users\Downloads
D:\Folder Pelajaran

Tidak ada batasan ukuran folder. Namun semakin besar ukuran folder, semakin lama waktu kompresi yang dibutuhkan.

Use case ini menghasilkan path file/folder yang nantinya akan digunakan oleh use case kompresi dan dekompresi.

- Menginput Path Folder

Merupakan salah satu cara bagi pengguna untuk menentukan file/folder yang ingin diproses. Pengguna akan mengetikkan path pada text field pada antarmuka CompactJV.

Contoh path yang diinput:

C:\Users\Documents

Path bisa mengarah ke file tunggal atau folder/direktori yang berisi banyak file di dalamnya. Path folder akan memproses semua file di dalam folder tersebut. Inputan path kemudian divalidasi untuk memastikan file/folder tersebut ada. Jika tidak valid, akan ditampilkan pesan kesalahan.

- Drag and Drop File/Folder

Cara lain untuk memilih file/folder adalah dengan melakukan drag and drop dari file explorer Windows ke dalam jendela aplikasi CompactJV. Pengguna tinggal drag file/folder yang ingin diproses, lalu melepaskan (drop) di area CompactJV. Drag and drop merupakan cara yang lebih cepat dan mudah dibanding mengetikkan path manual. Baik file tunggal atau folder bisa dipilih dengan cara ini. Hasilnya sama dengan use case menginput path secara manual.

- Memilih Algoritma Kompresi

Setelah file/folder dipilih, pengguna dapat memilih algoritma kompresi yang ingin digunakan untuk mengompresi file tersebut.

Pilihan algoritma mencakup:

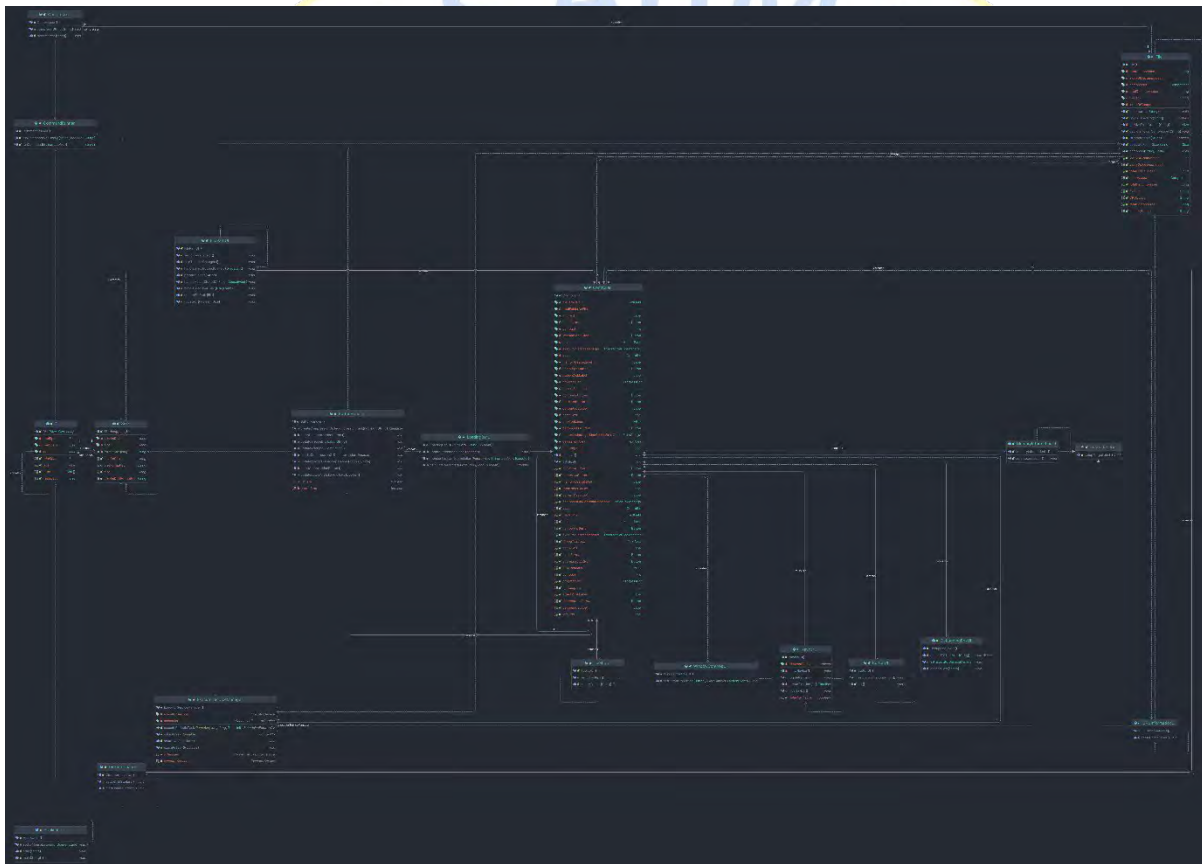
- XPRESS4K
- XPRESS8K
- XPRESS16K
- LZX

Masing-masing algoritma dijelaskan secara singkat pada antarmuka agar pengguna dapat memilih sesuai kebutuhan. Misalnya XPRESS4K untuk kecepatan, atau LZX untuk kompresi maksimum tapi lebih lambat. Pilihan ini nantinya akan digunakan saat menjalankan proses kompresi.

- **Mengompresi File/Folder**
Merupakan proses inti dari aplikasi CompactJV, yaitu mengompresi file atau folder yang telah dipilih pengguna menggunakan algoritma kompresi yang dipilih. Proses kompresi melibatkan:
 - Memanggil executable compact.exe dengan parameter path file, algoritma, dan flags
 - Menulis file terkompresi (.cmp) ke lokasi yang sama dengan aslinya
 - Menghapus file asli setelah berhasil dikompresi
 - Menampilkan status kompresi pada UIProses kompresi berjalan pada background thread agar UI tetap responsif. File asli tidak dihapus jika kompresi gagal. Hasil dari kompresi ini adalah file .cmp yang ukurannya lebih kecil dari aslinya.
- **Mendekompresi File**
Kebalikan dari kompresi, use case ini menerima file berekstensi .cmp yang terkompresi, lalu mengembalikannya ke bentuk asli dengan ukuran normal. Proses dekompresi juga melibatkan compact.exe.
- **Menampilkan Status Kompresi**
Saat proses kompresi atau dekompresi sedang berjalan, UI CompactJV akan menampilkan status berupa:
 - Persentase proses yang sudah selesai
 - Ukuran file saat ini
 - Estimasi waktu yang dibutuhkan
 - Kecepatan kompresi/dekompresi
 - Informasi ini didapat dengan membaca output dari compact.exe secara berkala.Tujuannya agar pengguna dapat memantau progres kompresi dan estimasi waktu penyelesaian.
- **Mengelola Resource Komputer**
Informasi ditampilkan pada antarmuka agar pengguna mengetahui resource apa yang sedang digunakan intensif. Data didapat dengan mengakses performance counters pada Windows. Selama proses kompresi berjalan yang melibatkan CPU dan I/O intensif, CompactJV juga memantau penggunaan resource komputer seperti:
 - CPU
 - Memori
 - Penyimpanan

Dengan demikian pengguna juga dapat memonitor sistemnya selama kompresi berjalan. Dari uraian use case di atas dapat disimpulkan bahwa CompactJV menyediakan fungsionalitas utama kompresi dan dekompresi file dengan performa yang optimal. Pengguna dilibatkan dalam menentukan file/folder sasaran beserta algoritma kompresi yang diinginkan. Antarmuka yang intuitif dan informatif memudahkan interaksi pengguna dengan sistem.

2.4. UML Class Diagram



Gambar 24 UML Class Diagram, CompactJV

Untuk melihat UML class Diagram lebih jelas, dapat melihat link berikut.

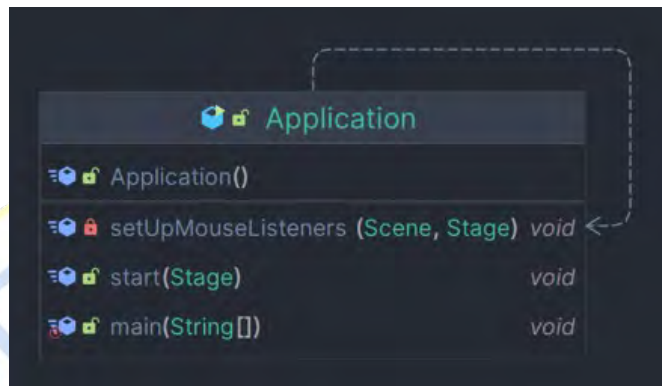
<https://github.com/IRedDragonICY/CompactJV/blob/main/documentation/class%20diagram.jpg>

Aplikasi CompactJV dibangun dengan menerapkan pola arsitektur perangkat lunak Model-View-Controller (MVC) yang memisahkan antara logika bisnis aplikasi (model), antarmuka pengguna (view), dan pengontrol (controller).

Pemisahan ini bertujuan untuk meningkatkan modularitas, reusability, dan maintainability kode dengan membagi komponen berdasarkan tanggung jawabnya masing-masing.

Berikut adalah penjelasan rinci mengenai peran dan detail implementasi dari setiap class pada diagram UML CompactJV:

2.4.1. Class Application



Gambar 25 class Application.java

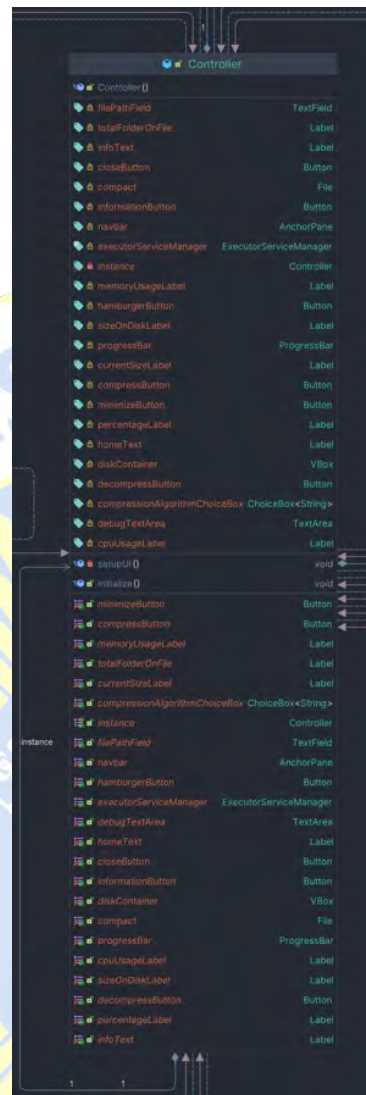
Class Application merupakan class utama (entry point) dari program CompactJV yang meng-extend Application class dari JavaFX. Class ini bertanggung jawab untuk:

- Memulai dan meluncurkan Graphical User Interface (GUI) aplikasi
- Membuat scene dan mengatur stage untuk tampilan
- Mengatur property window seperti ukuran, title, dan icon

Method start() akan dipanggil secara otomatis saat aplikasi diluncurkan. Di dalamnya, objek scene dibuat dengan meload file FXML layout menggunakan FXMLLoader. Scene tersebut kemudian ditampilkan pada stage. Pengaturan window seperti ukuran, judul, dan icon aplikasi juga dilakukan di dalam method start() ini.

Application hanya bertugas menampilkan GUI, tidak terlibat dengan logika bisnis aplikasi. Oleh karena itu peran class ini sesuai dengan konsep View pada arsitektur MVC. Application bertindak sebagai starting point dan container bagi komponen view lainnya. Namun logika program didelegasikan ke Controller dan Model sesuai prinsip separation of concerns pada MVC.

2.4.2. Class Controller



Gambar 26 Class Controller

Controller adalah class utama yang berperan sebagai pengontrol utama dalam arsitektur MVC pada aplikasi CompactJV.

Controller berfungsi sebagai jembatan penghubung antara View dan Model. Ia menerima input dari pengguna melalui View, kemudian memanggil fungsi bisnis yang tepat pada Model, lalu mengirimkan hasilnya kembali ke View untuk ditampilkan.

Tanggung jawab utama dari Controller adalah:

- Mengelola object model dan mapper
- Mengolah input dari pengguna
- Memanggil fungsi bisnis pada model
- Mengelola alur kerja aplikasi
- Berkomunikasi dengan View untuk menampilkan hasil

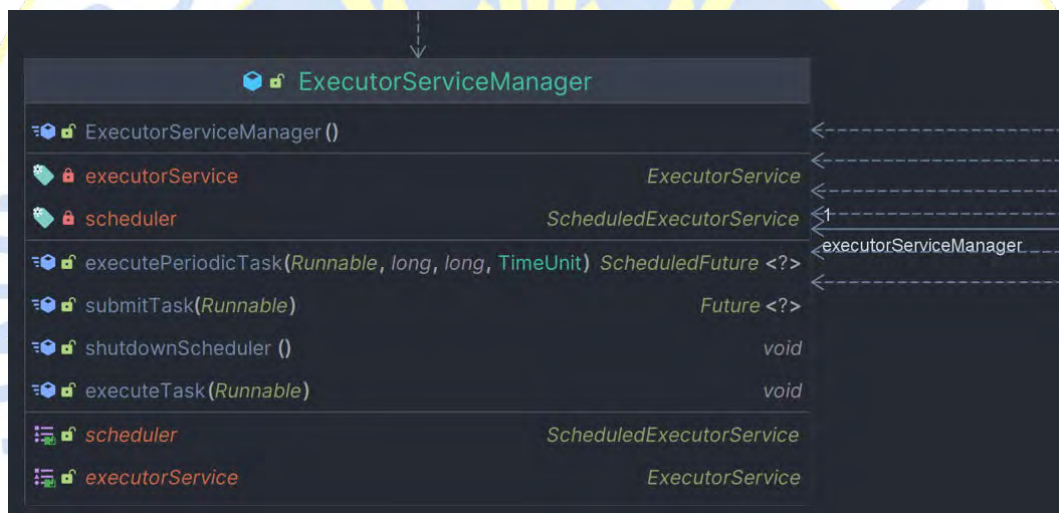
Pada implementasi di CompactJV, Controller digunakan untuk:

- Mengelola `ExecutorServiceManager` untuk eksekusi asynchronous
- Mengelola objek File sebagai model bisnis utama aplikasi
- Meng-inisialisasi komponen UI
- Mengatur interaksi dan alur data antar komponen

Controller memegang referensi ke semua objek UI agar bisa diakses dari satu tempat dan dikelola. Method `initialize()` akan dipanggil sekali saat Controller pertama kali dimuat untuk menginisialisasi komponen UI dan logika aplikasi.

Dengan penerapan pola MVC, ketergantungan dan coupling antar komponen bisa dikurangi. Controller berperan sebagai orchestrator tanpa terlibat secara langsung dengan detail implementasi model ataupun view.

2.4.3. Class `ExecutorServiceManager`



Gambar 27 Class `ExecutorServiceManager`

`ExecutorServiceManager` digunakan oleh Controller untuk mengelola eksekusi task secara asynchronous (multithreaded). Class ini membungkus `ExecutorService` dan `ScheduledExecutorService` dari `java.util.concurrent` package sehingga bisa diakses dengan mudah oleh Controller.

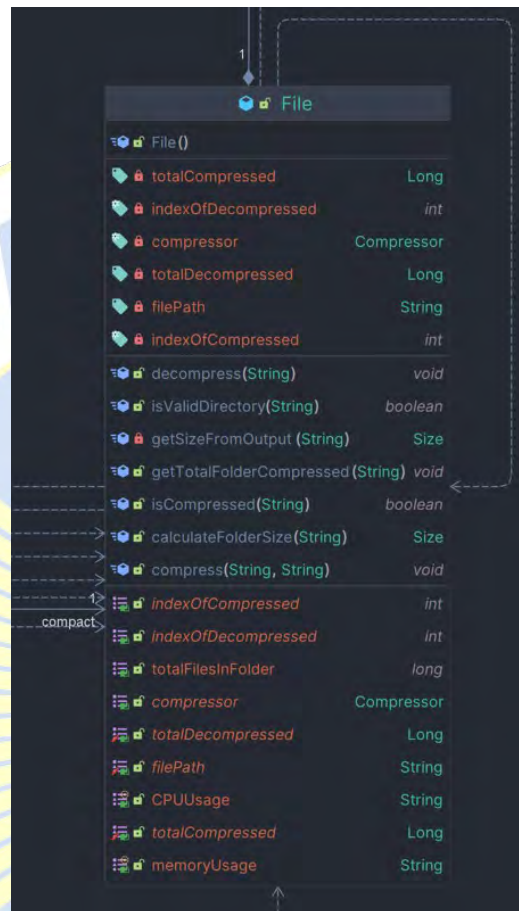
Fungsi utama dari `ExecutorServiceManager` adalah:

- Menyediakan thread pool untuk eksekusi asynchronous
- Menjalankan task sekali (`executeTask`)
- Menjalankan task berulang secara berkala (`executePeriodicTask`)
- Mengirimkan task yang mengembalikan nilai `Future` (`submitTask`)

Dengan adanya `ExecutorServiceManager`, operasi berat seperti kompresi file dapat dijalankan secara asynchronous sehingga tidak memblock thread utama dan UI. `ExecutorServiceManager` juga menyediakan `ScheduledExecutorService` yang bisa digunakan untuk menjadwalkan task secara berkala, misalnya untuk auto update data. Class

ini merupakan bagian dari controller yang membantu manajemen eksekusi task. Namun ia tidak terlibat secara langsung dengan view maupun model.

2.4.4. Class File



Gambar 28 Class File

File adalah model utama pada aplikasi CompactJV yang merepresentasikan file komputer yang akan dikompresi atau didekompresi. Class ini menyimpan informasi penting mengenai file seperti:

- Path lokasi file
- Ukuran file saat ini
- Ukuran file setelah dikompresi
- Apakah file sudah terkompresi atau belum
- Total file dan folder pada direktori

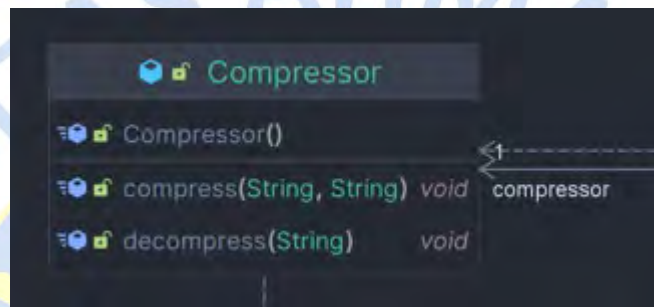
Selain itu File juga menyediakan fungsi-fungsi utama terkait kompresi file seperti:

- Kompresi file (compress)
- Dekompresi file (decompress)
- Menghitung total ukuran file pada direktori (calculateFolderSize)
- Validasi apakah direktori valid (isValidDirectory)

- Menghitung total file pada direktori (getTotalFilesInFolder)
- Dan lainnya

Fungsi-fungsi ini nantinya akan dipanggil oleh Controller saat diperlukan. File berinteraksi dengan Compressor untuk melakukan kompresi menggunakan compact.exe. Ukuran file di representasikan dalam objek Size. Jadi secara keseluruhan File berperan sebagai model bisnis inti pada arsitektur MVC aplikasi ini.

2.4.5. Class Compressor



Gambar 29 Class Compressor

Compressor adalah model yang bertanggung jawab melakukan kompresi dan dekompresi file dengan memanggil executable compact.exe. Ia menerima input berupa path file dan algoritma kompresi yang ingin digunakan. Kemudian akan mengeksekusi perintah compact melalui command line secara asynchronous.

Fungsi utamanya adalah:

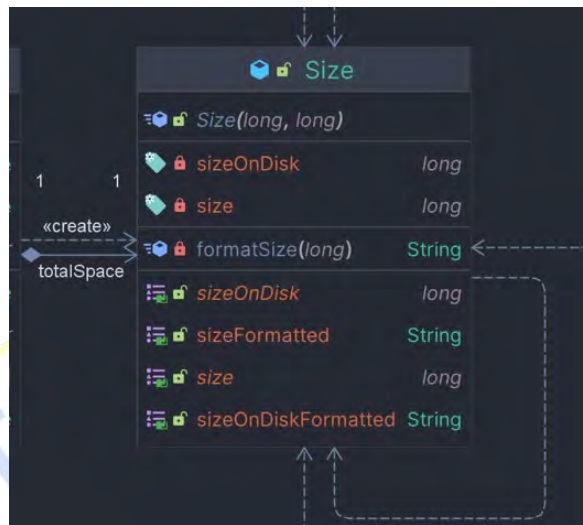
- Kompresi file (compress)
- Dekompresi file (decompress)

Compressor merupakan model murni tanpa referensi apapun ke view atau controller. Ia hanya berfokus pada enkapsulasi bisnis inti yaitu kompresi dan dekompresi.

Pemisahan ini sesuai prinsip pemisahan model pada arsitektur MVC dimana model hanya bertanggung jawab pada logika bisnis inti, tidak bergantung komponen lain.

Dengan adanya Compressor, logika kompresi bisa dengan mudah diganti dengan library compression lainnya tanpa mempengaruhi bagian view ataupun controller.

2.4.6. Class Size



Gambar 30 Class Size.

Size merupakan model sederhana yang digunakan untuk merepresentasikan ukuran file dalam satuan bytes.

Class ini menyimpan dua buah data:

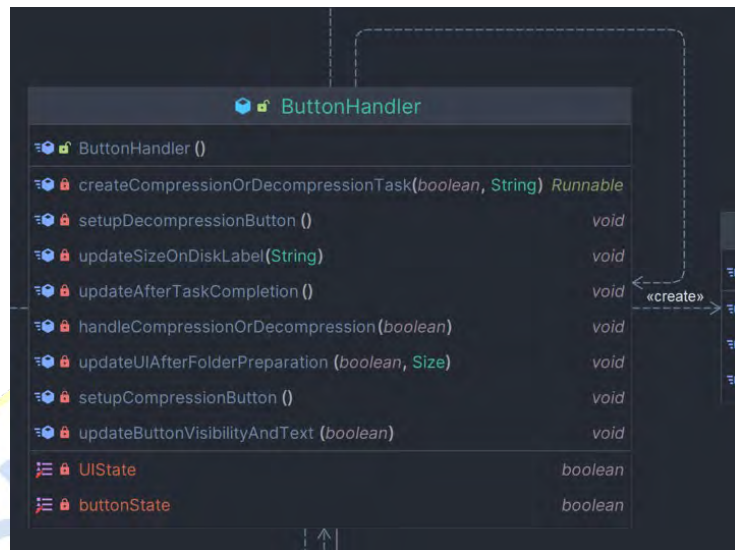
- ukuran asli file
- ukuran file setelah dikompresi

Keduanya disimpan dalam tipe data long yang menyimpan nilai bytes. Size juga menyediakan dua buah function untuk formatting ukuran ke dalam satuan yang lebih readable bagi user:

- getSizeFormatted() untuk ukuran asli
- getSizeOnDiskFormatted() untuk ukuran kompresi

Output formatting ini misalnya dalam satuan KB, MB atau GB tergantung total bytes-nya. Class File menggunakan objek dari model Size ini untuk menyimpan informasi ukuran file sebelum dan sesudah dikompresi.

2.4.7. Class ButtonHandler



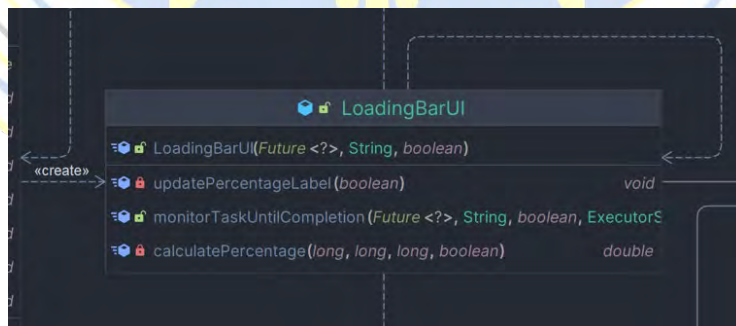
Gambar 31 Class ButtonHandler.

ButtonHandler merupakan controller khusus yang bertugas menangani event click pada tombol utama yaitu Compress dan Decompress. Ketika salah satu tombol diklik, ButtonHandler akan memanggil function handleCompressionOrDecompression() dengan parameter boolean isCompress untuk membedakan aksi kompresi dan dekompresi. Function ini kemudian akan mengeksekusi proses kompresi atau dekompresi secara asynchronous dengan memanggil Controller.

Jadi secara umum ButtonHandler hanya bertanggung jawab untuk menangani event klik pada dua tombol utama, tanpa terlibat dengan implementasi detailnya.

Pekerjaan ButtonHandler sesuai dengan konsep Controller pada MVC yang hanya mengkoordinasikan komunikasi tanpa terlibat secara langsung dengan model ataupun view.

2.4.8. Class LoadingBarUI



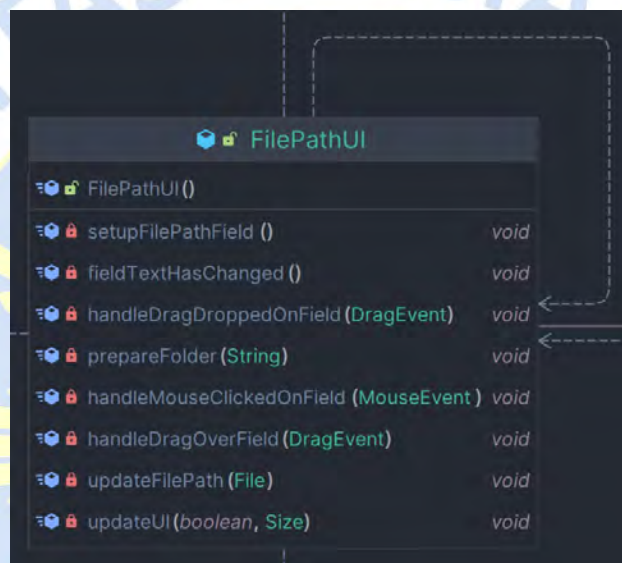
Gambar 32 Class LoadingBarUI.

LoadingBarUI merupakan view yang bertugas untuk menampilkan progress bar serta persentase proses kompresi atau dekompresi file. Ia menerima object Future dari task asynchronous yang dieksekusi Controller melalui ExecutorServiceManager. Dengan

memonitor state dari Future object ini, LoadingBarUI bisa menampilkan progress kompresi atau dekompresi secara real-time. Class ini berjalan pada thread yang berbeda agar tidak mengganggu operasi utama dari aplikasi.

LoadingBarUI hanya bertanggung jawab untuk menampilkan progress bar dan persentase kompresi. Logika bisnis tetap berada pada Controller dan Model. Fungsinya sesuai dengan View pada MVC yang hanya menampilkan antarmuka tanpa terlibat dengan logika inti aplikasi.

2.4.9. Class FilePathUI

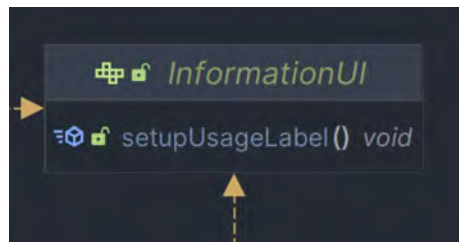


Gambar 3.3 Class FilePathUI.

FilePathUI merupakan view controller yang khusus menangani text field input path file pada UI CompactJV. Tugas utamanya adalah menangani event-event seperti click, drag and drop pada text field input path agar bisa menerima input dari pengguna.

Selain itu FilePathUI juga melakukan validasi terhadap isi text field. Jika input path tidak valid, maka akan ditampilkan pesan error. Namun jika valid, FilePathUI akan memanggil Controller untuk mempersiapkan folder sebelum dilakukan kompresi atau dekompresi. Jadi secara garis besar FilePathUI hanya bertanggung jawab terhadap text field input path saja, tanpa terlibat dengan implementasi logika bisnis aplikasi. Peran ini sesuai dengan View pada arsitektur MVC yang hanya bertugas menangani interaksi pengguna dan antarmuka.

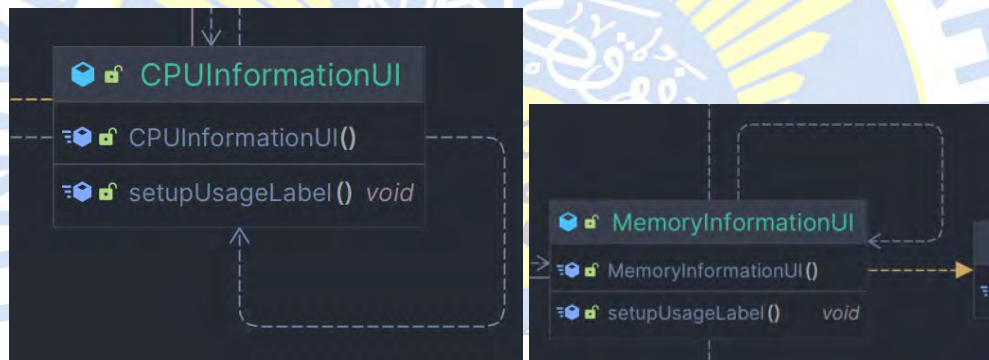
2.4.10. Interface InformationUI



Gambar 34 Interface InformationUI.

`InformationUI` merupakan interface yang digunakan untuk standarisasi cara kerja UI untuk menampilkan informasi penggunaan resource. Interface ini hanya memiliki satu method abstract yaitu `setupUsageLabel()` yang wajib diimplementasikan oleh class turunannya. Dengan adanya standarisasi interface ini, pengembangan fitur informasi resource menjadi lebih terstruktur dan konsisten. Interface `InformationUI` juga menerapkan prinsip program ke interface bukan implementasi konkritnya pada Java.

2.4.11. Class `CPUInformationUI` dan `MemoryInformationUI`



Gambar 35 Class `CPUInformationUI` dan `MemoryInformationUI`.

`CPUInformationUI` dan `MemoryInformationUI` adalah dua buah UI controller yang mengimplementasikan `InformationUI` interface. Keduanya bertugas untuk menampilkan informasi penggunaan resource CPU dan memori secara real-time dengan meng-override method `setupUsageLabel()`. Informasi usage didapatkan dan diperbarui secara berkala menggunakan task asynchronous melalui `ExecutorServiceManager`. Kedua class ini hanya bertanggung jawab untuk menampilkan informasi penggunaan resource yang sesuai dengan namanya. Sama seperti view lainnya, ia tidak terlibat dengan logika bisnis inti aplikasi.

Dengan demikian seluruh class pada diagram UML Class Diagram CompactJV telah mengimplementasikan prinsip-prinsip penting dalam rekayasa perangkat lunak seperti:

- Encapsulation
- Abstraction
- Separation of concerns
- Program to interfaces

- Single responsibility
- High cohesion, loose coupling

Ini memungkinkan modularitas dan maintainability kode yang sangat baik karena setiap class dan komponen memiliki tanggung jawab dan peran yang jelas.

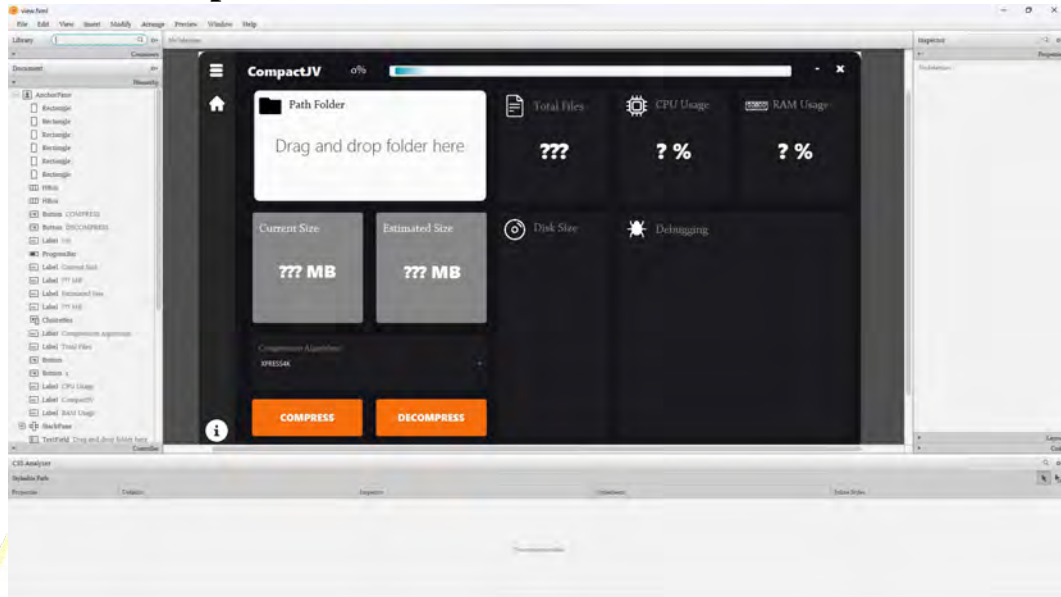
Semua ketergantungan antar class telah diminimalkan sehingga class bisa berubah atau diganti tanpa berpengaruh pada bagian lainnya. Misalnya jika ingin mengganti library kompresi, cukup modifikasi Compressor class tanpa menyentuh UI ataupun Controller. Begitu pula jika ingin menambah fitur notifikasi, cukup buat class NotificationUI tanpa harus mengubah model dan controller lainnya.

Penerapan pola MVC memastikan pemisahan yang jelas antara view yang menampilkan UI, controller yang mengatur workflow, dan model yang melakukan logika bisnis inti aplikasi. Dispatcher Servlet pada framework Spring MVC merupakan contoh konkrit controller yang powerful untuk aplikasi enterprise. Sedangkan Repository dan Service class pada Spring bertindak sebagai model. Dan templating engine seperti Thymeleaf sebagai viewnya. Jadi konsep MVC adalah pola arsitektur yang sangat penting dalam pengembangan aplikasi besar maupun kecil. Dibandingkan gaya spaghetti code, MVC membuat kode jauh lebih terstruktur, reusable, dan maintainable. MVC juga mendorong peer programmer untuk fokus pada satu area tertentu, misalnya frontend developer bisa fokus pada view. Sedangkan backend developer mengerjakan model tanpa perlu terlibat UI. Lalu controller menjadi jembatan penghubung di antara keduanya. Ini memungkinkan tim developer bekerja secara paralel dan independent tanpa mengganggu area lainnya.

Jadi kesimpulannya, penerapan pola MVC yang konsisten sangat disarankan untuk aplikasi apapun baik kecil maupun enterprise. Karena manfaatnya yang signifikan dalam membuat struktur dan alur kode menjadi jauh lebih baik dan profesional.

2.5. Design Interface

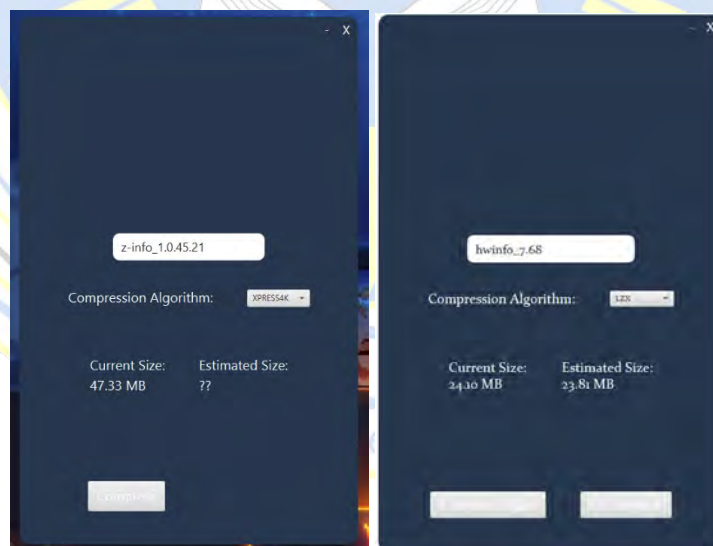
2.5.1. Deskripsi Umum



Gambar 36 Tampilan scene builder

Design interface CompactJV dirancang dengan menggunakan Scene Builder, sebuah alat pengembangan GUI untuk aplikasi JavaFX. Desain ini memberikan antarmuka pengguna yang intuitif dan responsif, memudahkan pengguna untuk melakukan kompresi dan dekompresi file dengan mudah.

2.5.2. Riwayat Design CompactJV



Gambar 37 Desain CompactJV pada versi Alpha v0.1 sebagai prototype.

Antarmuka pengguna ini adalah sebuah prototype yang dirancang dengan menggunakan JavaFX. Struktur utama dari antarmuka ini diatur dalam AnchorPane,

sebuah kontainer yang memungkinkan penempatan bebas elemen-elemen menggunakan koordinat absolut.

Di bagian paling atas, terdapat dua tombol penting yang diletakkan di pojok kanan atas, yaitu closeButton dan minimizeButton. closeButton berfungsi untuk menutup aplikasi, berperan penting dalam memberikan kontrol kepada pengguna untuk mengakhiri sesi penggunaan aplikasi kapan pun mereka inginkan. Sementara itu, minimizeButton berfungsi untuk meminimalkan ukuran jendela aplikasi, memberikan fleksibilitas kepada pengguna dalam mengelola ruang kerja mereka.

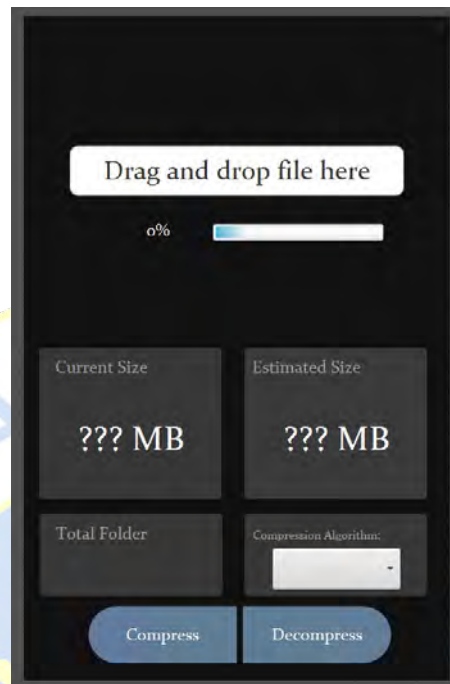
Pada bagian tengah, terdapat sebuah TextField dengan id "filePathField". TextField ini diletakkan di tengah AnchorPane, memberikan posisi yang mudah diakses oleh mata dan cursor pengguna. TextField ini digunakan untuk memasukkan jalur file yang akan dikompresi atau didekompresi. Desain ini bertujuan untuk memudahkan pengguna dalam menginput file yang mereka inginkan.

Di bawah TextField, terdapat dua pasangan label dan teks yang diposisikan secara horisontal. Label pertama bertuliskan "Current Size:", bertujuan untuk memberikan informasi kepada pengguna tentang ukuran file saat ini. Pasangan label dan teks kedua bertuliskan "Estimated Size:", memberikan perkiraan ukuran file setelah dilakukan kompresi atau dekompresi. Fitur ini sangat berguna bagi pengguna untuk memahami seberapa efektif proses kompresi atau dekompresi yang mereka lakukan.

Selanjutnya, di bawah kedua pasangan label dan teks tersebut, terdapat sebuah label dan ChoiceBox. Label ini bertuliskan "Compression Algorithm:", memberikan konteks kepada ChoiceBox yang berisi pilihan algoritma kompresi yang dapat digunakan pengguna. ChoiceBox ini memberikan kebebasan kepada pengguna untuk memilih algoritma kompresi sesuai kebutuhan mereka.

Terakhir, di bagian bawah AnchorPane, terdapat dua tombol: compressButton dan decompressButton. Tombol compressButton digunakan untuk memulai operasi kompresi, sementara tombol decompressButton digunakan untuk memulai operasi dekompresi. Penempatan tombol ini di bagian bawah memudahkan akses pengguna dan memberikan flow yang intuitif dalam pengoperasian aplikasi.

Secara keseluruhan, antarmuka pengguna ini dirancang dengan pemosisian yang rapi dan intuitif, memudahkan pengguna dalam mengoperasikan aplikasi ini. Prototype ini berperan penting dalam proses pengujian dan evaluasi awal sebelum pengembangan lebih lanjut, serta memberikan gambaran awal tentang bagaimana aplikasi ini akan bekerja dan digunakan.



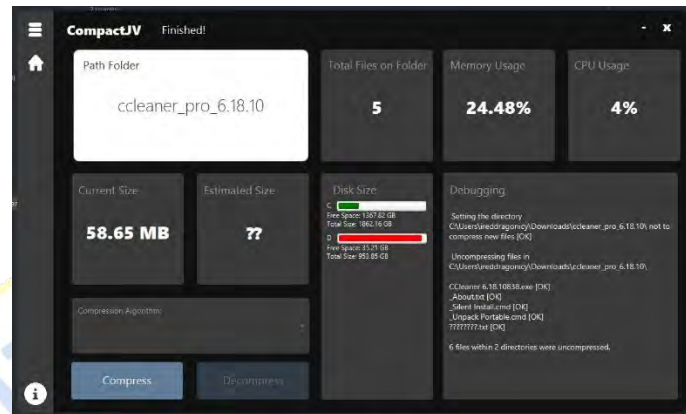
Gambar 38 Mencoba berpindah dari prototype menjadi desain modern vertical app desktop.

Desain antarmuka pengguna yang di atas menggunakan JavaFX dan XML, menawarkan estetika modern dan minimalis. Dibandingkan dengan desain sebelumnya, desain ini menunjukkan peningkatan penggunaan elemen kotak-kotak, memberikan tampilan yang lebih terstruktur dan terorganisir. Kontainer utama, yang dikenal sebagai `AnchorPane`, menampung semua elemen lainnya, membentuk kerangka kerja tata letak yang serbaguna.

Elemen-elemen seperti `Rectangle` digunakan untuk memberikan latar belakang visual yang menarik untuk beberapa komponen, sementara `TextField` memfasilitasi masukan teks, dalam hal ini, path file yang ingin dikompres atau didekompres. Tombol, seperti "Compress" dan "Decompress", memulai proses kompresi atau dekompresi, dan Label digunakan untuk menampilkan informasi relevan, seperti "Current Size", "Estimated Size", dan "Compression Algorithm".

Selain itu, `ProgressBar` menampilkan perkembangan proses kompresi atau dekompresi, dan `ChoiceBox` memberikan dropdown menu yang memungkinkan pengguna memilih algoritma kompresi yang mereka inginkan. Dibandingkan dengan desain sebelumnya, desain ini tampaknya lebih intuitif dan user-friendly, memungkinkan pengguna untuk menavigasi dan berinteraksi dengan aplikasi dengan lebih mudah.

Secara keseluruhan, desain ini berhasil menggabungkan estetika modern dan fungsionalitas tinggi, menciptakan pengalaman pengguna yang meningkat dibandingkan dengan desain sebelumnya.



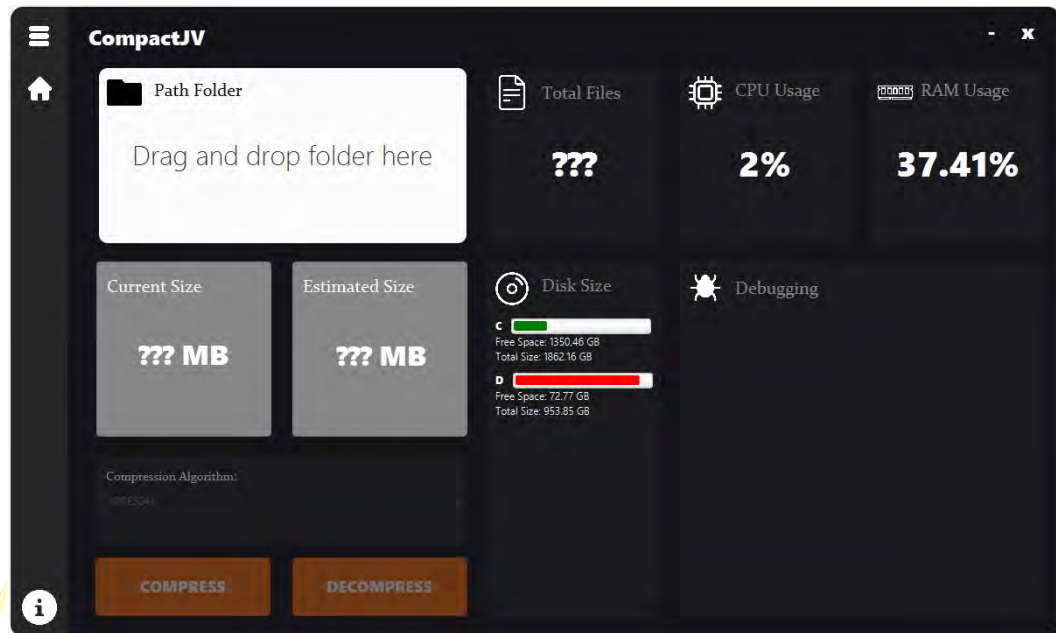
Gambar 39 Modern GUI CompactJV berganti dari vertical menjadi horizontal.

Desain FXML awal dari CompactJV menunjukkan tata letak vertikal dengan elemen-elemen seperti tombol kompresi, dekompresi, label persentase, progress bar, dan pilihan algoritma kompresi. Pengaturan komponen tersebut terlihat lebih simpel dan terfokus pada fungsionalitas dasar aplikasi.

Desain FXML telah diperbarui untuk mencapai tata letak horizontal yang lebih terorganisir. Perubahan ini juga melibatkan penambahan elemen-elemen baru seperti navigasi bar di sisi kiri. Berikut adalah beberapa perubahan utama seperti, Navbar ditambahkan di sisi kiri antarmuka untuk meningkatkan navigasi dan memberikan tampilan yang lebih terstruktur. Komponen-komponen utama seperti tombol kompresi, dekompresi, label persentase, progress bar, dan pilihan algoritma kompresi ditempatkan secara horizontal untuk memperbaiki tata letak dan membuatnya lebih intuitif. Beberapa elemen, seperti ukuran folder total dan label penggunaan CPU, dipindahkan ke lokasi yang lebih sesuai untuk menciptakan keseimbangan visual. Tombol minimize dan close pada bagian atas kanan sekarang memiliki tooltip yang memberikan petunjuk visual tambahan

Perubahan desain ini dilakukan untuk meningkatkan keterbacaan antarmuka dan menyederhanakan navigasi bagi pengguna. Tata letak horizontal memberikan tampilan yang lebih terstruktur, sementara penambahan navbar memberikan akses cepat ke fitur-fitur utama aplikasi.

Desain CompactJV yang telah diperbarui menawarkan tampilan yang lebih terorganisir dan efisien. Pengguna dapat dengan mudah mengakses fungsi utama aplikasi dengan bantuan navbar, sementara antarmuka yang lebih horizontal memberikan pengalaman pengguna yang lebih intuitif. Perubahan ini diharapkan dapat meningkatkan penggunaan dan estetika keseluruhan dari CompactJV.



Gambar 40 Final design CompactJV

Dalam desain terakhir, perubahan signifikan telah dilakukan untuk meningkatkan tampilan dan fungsionalitas aplikasi "CompactJV". Warna keseluruhan antarmuka telah diubah menjadi #151419, memberikan tampilan yang lebih gelap, modern, dan elegan dibandingkan dengan desain sebelumnya yang lebih cerah. Penggunaan elemen Rectangle pada berbagai bagian, seperti EstimatedSizeRectangle dan CurrentSizeRectangle, menambahkan sentuhan estetis yang menyatu dengan tema keseluruhan.

Tombol COMPRESS dan DECOMPRESS memiliki peningkatan signifikan dalam hal penataan warna dan font, menggunakan kombinasi oranye dan putih dengan font yang lebih besar, sehingga meningkatkan keterbacaan. Penambahan ProgressBar memberikan feedback visual yang jelas terkait kemajuan proses kompresi atau dekompresi kepada pengguna.

ChoiceBox untuk memilih algoritma kompresi diperkenalkan dengan warna latar belakang yang serasi, memberikan pengguna lebih banyak kontrol atas proses tersebut. Icon pada elemen-elemen seperti ImageView menambahkan elemen visual yang sesuai dengan konteks, seperti ikon folder, ikon CPU, dan lainnya.

Penambahan elemen ScrollPane dan TextArea untuk debugging memudahkan pengguna untuk melihat log atau pesan debug dengan jelas dan terorganisir. Navbar di sebelah kiri, termasuk tombol seperti informationButton, homeButton, dan hamburgerButton, memberikan navigasi yang lebih intuitif bagi pengguna.

Secara keseluruhan, desain terakhir tidak hanya meningkatkan tampilan visual dan estetika antarmuka, tetapi juga menambahkan elemen-elemen fungsional yang memperkaya pengalaman pengguna, seperti ProgressBar dan area debugging. Perubahan ini diharapkan dapat meningkatkan daya tarik dan kenyamanan pengguna dalam menggunakan aplikasi "CompactJV".

2.5.3. Komponen Utama

Rectangle:

EstimatedSizeRectangle dan CurrentSizeRectangle: Menampilkan grafis persegi panjang untuk menunjukkan ukuran estimasi dan ukuran aktual dari file yang akan dikompresi.

CompressionAlgorithmRectangle: Menunjukkan pilihan algoritma kompresi yang dipilih oleh pengguna.

TotalFilesRectangle, CPUUsageRectangle, dan RAMUsageRectangle: Memberikan tampilan visual untuk informasi jumlah total file, penggunaan CPU, dan penggunaan RAM.

DiskSizeRectangle dan DebuggingRectangle: Memberikan tampilan visual untuk informasi ukuran disk dan debugging.

Buttons:

compressButton dan decompressButton: Tombol untuk memulai proses kompresi dan dekompresi.

minimizeButton dan closeButton: Tombol untuk meminimalkan dan menutup aplikasi.

Labels:

percentageLabel: Menampilkan persentase progres saat proses kompresi atau dekompresi berlangsung.

currentSizeLabel dan sizeOnDiskLabel: Menampilkan ukuran aktual dan estimasi ukuran file yang akan dikompresi.

compressionAlgorithmLabel: Menampilkan label untuk memilih algoritma kompresi.

cpuUsageLabel dan memoryUsageLabel: Menampilkan persentase penggunaan CPU dan RAM.

totalFolderOnFile: Menampilkan jumlah total folder pada file yang akan dikompresi.

ChoiceBox:

compressionAlgorithmChoiceBox: Pilihan dropdown untuk memilih algoritma kompresi yang diinginkan.

ProgressBar:

progressBar: Menampilkan progres visual saat proses kompresi atau dekompresi berlangsung.

TextField:

filePathField: Menampilkan path folder yang akan dikompresi, dan mendukung fitur drag-and-drop.

ImageView:

Berbagai ikon: Menunjukkan ikon-ikon untuk memberikan representasi visual terhadap beberapa elemen, seperti jenis file, CPU, RAM, dll.

StackPane dan VBox:

diskContainer: Menampilkan informasi ukuran disk dalam bentuk persegi panjang di dalam StackPane.

AnchorPane:

navbar: Bagian sidebar aplikasi yang memuat beberapa tombol navigasi, seperti Home dan Information.

ScrollPane dan TextArea:

debugScrollPane dan debugTextArea: Menampilkan informasi debugging dalam bentuk scrollable text area.

Desain Warna dan Font

Warna-warna yang digunakan, seperti hitam, putih, dan oranye, memberikan kontras yang baik dan membuat teks serta elemen lain mudah terbaca. Penggunaan font Segoe UI dan variasi ukuran font memberikan tampilan yang modern dan bersih.

2.6. Code

CompactJV adalah alat kompresi yang kuat yang dikembangkan menggunakan Java. Alat ini memanfaatkan kemampuan dari Compact.exe dari Windows SDK untuk mengompres file seperti game atau aplikasi. Dirancang dengan pengguna sistem operasi Windows ini, CompactJV menawarkan kompresi file dengan kinerja tinggi dan mudah digunakan.

Berikut adalah beberapa langkah dasar yang biasanya dilakukan oleh aplikasi ini:

- Pengguna memilih file yang ingin dikompres.
- Aplikasi ini kemudian menggunakan Compact.exe untuk mengompres file tersebut.
- Setelah proses kompresi selesai, pengguna dapat menyimpan file yang telah dikompres atau menggunakannya sesuai kebutuhan mereka.

Dalam proses pembuatan aplikasi JavaFX, kita dimulai dengan definisi beberapa konstanta yang akan digunakan nantinya. Konstanta ini mencakup judul aplikasi, path ikon, dan path view. Selain itu, dua variabel tambahan juga didefinisikan untuk mengatur posisi window saat pengguna melakukan operasi drag.

Setelah konstanta dan variabel didefinisikan, kita mulai dengan metode utama yang menjadi titik awal eksekusi aplikasi. Di dalam metode ini, kita melakukan serangkaian operasi untuk mengatur tampilan dan perilaku window aplikasi. Operasi ini termasuk pengambilan resource dari file FXML, pembuatan objek Scene, pengaturan style dan warna latar belakang, penambahan listener mouse, dan pengaturan judul dan ikon window.

Selanjutnya, kita menambahkan listener mouse ke scene untuk memungkinkan pengguna untuk menggeser window. Setelah semua pengaturan selesai, kita menampilkan scene di dalam window.

Akhirnya, kita memulai aplikasi dengan memanggil metode launch(). Ini adalah titik awal eksekusi program.

2.6.1. Class Button Handler

```
1 package com.example.compactjv.ui;
2
3 import com.example.compactjv.controller;
4 import com.example.compactjv.file;
5 import java.awt.application.platform;
6
7 import java.util.concurrent.Future;
8
9 public class ButtonHandler extends Controller {
10
11     public ButtonHandler() {
12         super();
13         setupCompressionButton();
14         setupDecompressionButton();
15     }
16
17     private void setupCompressionButton() {
18         compressButton.setOnAction(event -> handleCompressionOrDecompression(true));
19         setButtonState(true);
20     }
21
22     private void setupDecompressionButton() {
23         decompressButton.setOnAction(event -> handleCompressionOrDecompression(false));
24         setButtonState(true);
25     }
26
27     private void handleCompressionOrDecompression(boolean isCompression) {
28         String filePath = compact.getFilePath();
29         updateSizeOnDiskLabel("Loading...");
30         setButtonState(true);
31         Runnable task = createCompressionOrDecompressionTask(isCompression, filePath);
32         Future<> future = executorServiceManager.submitTask(task);
33         new LoadingBarUI(future, filePath, isCompression);
34     }
35
36     private Runnable createCompressionOrDecompressionTask(boolean isCompression, String filePath) {
37         return () -> {
38             progressBar.setVisible(true);
39             setUIState(true);
40             if (isCompression) {
41                 String algorithm = compressionAlgorithmChoiceBox.getValue();
42                 compact.compress(filePath, algorithm);
43             } else {
44                 compact.decompress(filePath);
45             }
46             updateAfterTaskCompletion();
47         };
48     }
49
50     private void updateAfterTaskCompletion() {
51         Size size = compact.calculateFolderSize(compact.getFilePath());
52         boolean isCompressed = compact.isCompressed(compact.getFilePath());
53         updateUIAfterFolderPreparation(isCompressed, size);
54         setButtonState(false);
55     }
56
57     private void updateSizeOnDiskLabel(String text) {
58         sizeOnDiskLabel.setText(text);
59     }
60
61     private void setButtonState(boolean state) {
62         compressButton.setDisable(state);
63         decompressButton.setDisable(state);
64     }
65
66     private void setUIState(boolean state) {
67         filePathField.setDisable(state);
68         compressionAlgorithmChoiceBox.setDisable(state);
69     }
70
71     private void updateUIAfterFolderPreparation(boolean isCompressed, Size size) {
72         Platform.runLater(() -> {
73             updateButtonVisibilityAndText(isCompressed);
74             currentSizeLabel.setText(size.getSizeFormatted());
75             setUIState(false);
76             sizeOnDiskLabel.setText(isCompressed ? size.getSizeOnDiskFormatted() : "");
77         });
78     }
79
80     private void updateButtonVisibilityAndText(boolean isCompressed) {
81         decompressButton.setDisable(!isCompressed);
82         compressButton.setText(isCompressed ? "Compress Again" : "Compress");
83     }
84 }
85
```

Gambar 41 Class Button Handler

Kelas `ButtonHandler` dalam kode yang telah di buat merupakan subclass dari `Controller`. Kelas ini bertugas untuk mengendalikan perilaku tombol-tombol dalam aplikasi. Berikut adalah penjelasan lebih detail tentang kelas ini:

- Konstruktor `ButtonHandler()` dipanggil saat instansiasi kelas. Konstruktor ini memanggil metode `setupCompressionButton()` dan `setupDecompressionButton()`, yang masing-masing mengatur perilaku tombol kompresi dan dekompresi.
- Metode `setupCompressionButton()` dan `setupDecompressionButton()` mengatur event handler untuk tombol kompresi dan dekompresi. Event handler ini memanggil metode `handleCompressionOrDecompression()` dengan parameter yang sesuai. Setelah itu, mereka menyetel status tombol menjadi aktif.
- Metode `handleCompressionOrDecompression()` menerima parameter boolean `isCompression` yang menentukan jenis operasi yang harus dilakukan (kompresi atau dekompresi). Metode ini memperbarui label `sizeOnDiskLabel`, menyetel status tombol menjadi aktif, membuat tugas kompresi atau dekompresi, dan menjalankan tugas tersebut secara asinkron. Selain itu, metode ini juga membuat UI loading bar.
- Metode `createCompressionOrDecompressionTask()` membuat tugas kompresi atau dekompresi berdasarkan parameter yang diberikan. Tugas ini mengatur visibilitas progress bar, menyetel status UI menjadi aktif, melakukan kompresi atau dekompresi, dan memperbarui UI setelah tugas selesai.
- Metode `updateAfterTaskCompletion()` menghitung ukuran folder dan mengecek apakah folder tersebut telah dikompresi. Kemudian, metode ini memperbarui UI setelah folder siap.
- Metode `setButtonState()` dan `setUIState()` digunakan untuk menyetel status tombol dan UI. Metode `updateSizeOnDiskLabel()` digunakan untuk memperbarui teks pada label `sizeOnDiskLabel`.

2.6.2. Class Button UI

[illegible]

Gambar 42 class button UI.

Kelas **ButtonUI** dalam kode merupakan bagian dari antarmuka pengguna (UI) dalam aplikasi. Berikut adalah penjelasan lebih rinci mengenai kelas ini:

1. Ekstensi dan Konstruktor:

Kelas **ButtonUI** adalah subclass dari kelas **Controller**, yang menunjukkan adanya pewarisan atau penggunaan fungsionalitas dari kelas induk. Konstruktor **ButtonUI()**

dipanggil saat membuat instans dari kelas ini. Konstruktor tersebut kemudian memanggil metode **init()** untuk melakukan inisialisasi awal.

2. Metode **setupInformationButton()**:

- Metode ini memiliki tanggung jawab utama dalam mengatur perilaku tombol informasi. Pada saat tombol informasi ditekan, sebuah dialog akan muncul. Dialog tersebut diatur untuk menjadi modul dan transparan.
- Dialog ini berisi elemen-elemen seperti judul, versi, logo, deskripsi, daftar pengembang, serta tombol "OK" dan "GitHub". Tombol "OK" berfungsi untuk menutup dialog, sementara tombol "GitHub" membuka halaman GitHub aplikasi dalam browser pengguna.
- Seluruh tata letak dan komponen-komponen tersebut diatur dengan menggunakan kelas-kelas JavaFX seperti **Stage**, **GridPane**, **Text**, **ImageView**, dan **Button**.

3. Metode **init()**:

- Metode **init()** bertujuan untuk melakukan inisialisasi kelas. Pada implementasi ini, metode tersebut memanggil **setupInformationButton()** untuk mengonfigurasi tombol informasi.

4. Penanganan Event:

- Dalam metode **setupInformationButton()**, diberikan penanganan event untuk aksi saat tombol informasi ditekan. Event handler ini menangkap aksi pengguna dan menentukan tindakan yang diambil, yaitu menampilkan dialog informasi.

5. Styling dan Tampilan UI:

- Dalam pembuatan dialog, diatur style dan tampilan elemen-elemen seperti teks dan tombol untuk memberikan antarmuka yang estetik dan sesuai dengan desain aplikasi.

6. Fungsi Tambahan:

- Kelas ini juga memiliki fungsi tambahan, seperti membuka halaman GitHub aplikasi ketika tombol "GitHub" ditekan. Ini memberikan keterlibatan pengguna lebih lanjut dengan proyek.

2.6.3. Class CPU Information



```
1 package com.example.compactjv.UI;
2
3 import com.example.compactjv.Controller;
4 import com.example.compactjv.File;
5 import javafx.application.Platform;
6
7 import java.util.concurrent.TimeUnit;
8
9 public class CPUInformationUI extends Controller implements InformationUI {
10
11     public CPUInformationUI() {
12         super();
13         setupUsageLabel();
14     }
15
16     @Override
17     public void setupUsageLabel() {
18         Runnable task = () -> {
19             String cpuUsage = File.getCPUUsage();
20             Platform.runLater(() -> cpuUsageLabel.setText(cpuUsage + "%"));
21         };
22
23         executorServiceManager.executePeriodicTask(task, 0, 1, TimeUnit.SECONDS);
24     }
25 }
```

Gambar 43 class CPUInformationUI.

Kode di atas adalah implementasi dari kelas CPUInformationUI dalam bahasa pemrograman Java dengan menggunakan JavaFX. Berikut adalah penjelasan singkat mengenai kode tersebut:


- Kelas CPUInformationUI merupakan subclass dari kelas Controller dan mengimplementasikan interface InformationUI. Kelas ini mungkin berperan sebagai bagian dari antarmuka pengguna untuk menampilkan informasi tentang penggunaan CPU.
- Konstruktor CPUInformationUI(): Konstruktor ini dipanggil saat membuat instansi dari kelas CPUInformationUI. Konstruktor ini memanggil konstruktor kelas induk (Controller) dan metode setupUsageLabel().
- Metode setupUsageLabel(): Metode ini diimplementasikan dari interface InformationUI. Dalam metode ini, sebuah Runnable dibuat untuk mendapatkan penggunaan CPU menggunakan metode statis File.getCPUUsage(). Penggunaan CPU kemudian diperbarui pada label (cpuUsageLabel) menggunakan Platform.runLater(), yang memastikan bahwa perubahan UI dilakukan di thread JavaFX. executorServiceManager digunakan untuk menjalankan tugas tersebut secara berkala setiap detik.

Kelas File digunakan untuk mendapatkan informasi tentang penggunaan CPU, dan kodenya mungkin terdapat pada metode statis getCPUUsage().

Runnable task berisi operasi untuk mendapatkan dan memperbarui penggunaan CPU. Platform.runLater() memastikan bahwa pembaruan UI dilakukan di thread JavaFX untuk menghindari konflik threading. executorServiceManager.executePeriodicTask() digunakan untuk menjalankan tugas tersebut secara berkala setiap detik, dengan menggunakan manajer eksekutor (executorServiceManager).

Secara keseluruhan, kelas `CPUInformationUI` sepertinya didesain untuk menampilkan informasi penggunaan CPU secara berkala pada suatu label dalam antarmuka pengguna JavaFX. Perhatikan bahwa implementasi lebih lanjut mungkin tergantung pada detail dari kelas-kelas yang tidak terlihat di dalam kode ini, seperti definisi `File` atau `executorServiceManager`.

2.6.4. Class Debugging Box

The image shows a screenshot of a code editor with a dark background and light-colored text. The code is in Java and defines a class named `DebuggingBoxUI` that extends `Controller`. The code includes package imports, class initialization, and methods for handling debugging output. The code is as follows:

```
1 package com.example.compactjv.ui;
2
3 import com.example.compactjv.Controller;
4 import com.example.compactjv.ExecutorServiceManager;
5 import javafx.application.Platform;
6
7 import java.io.ByteArrayOutputStream;
8 import java.io.OutputStream;
9 import java.io.PrintStream;
10
11 public class DebuggingBoxUI extends Controller {
12     private static final int MAX_TEXT_AREA_LENGTH = 5000;
13
14     public DebuggingBoxUI() {
15         super();
16         debugTextArea.setFont(javafx.scene.text.Font.font("Consolas"));
17         init(executorServiceManager);
18     }
19
20     private void init(ExecutorServiceManager executorServiceManager) {
21         executorServiceManager.executeTask(() -> {
22             OutputStream out = new ByteArrayOutputStream();
23             @Override
24             public void write(byte[] b, int off, int len) {
25                 appendToTextArea(new String(b, off, len));
26             }
27         });
28         System.setOut(new PrintStream(out, true));
29         System.setErr(new PrintStream(out, true));
30     }
31
32     private void appendToTextArea(final String str) {
33         Platform.runLater(() -> {
34             appendText(str);
35             debugTextArea.setScrollTop(Double.MAX_VALUE);
36         });
37     }
38
39     private void appendText(String str) {
40         String currentText = debugTextArea.getText();
41         debugTextArea.setText(currentText.length() > MAX_TEXT_AREA_LENGTH ? currentText.substring(0, MAX_TEXT_AREA_LENGTH) : currentText);
42         debugTextArea.appendText(str);
43     }
44 }
45
46 }
```

Gambar 44 class `DebuggingBox`.

Kelas `DebuggingBoxUI` dalam kode tersebut bertanggung jawab untuk menangkap dan menampilkan output debugging pada antarmuka pengguna JavaFX. Berikut adalah beberapa poin penting dalam kodenya:

1. Konstruktor dan Inisialisasi:

- Konstruktor `DebuggingBoxUI()` memanggil metode `init(executorServiceManager)` untuk melakukan inisialisasi awal.
- Dalam konstruktor, jenis font area teks (`debugTextArea`) diatur sebagai "Consolas".

2. Metode `init(ExecutorServiceManager executorServiceManager)`:

- Metode ini mengonfigurasi `System.out` dan `System.err` agar outputnya dapat ditangkap dan ditampilkan pada `debugTextArea`.
- Menggunakan `ByteArrayOutputStream` untuk menangkap output ke dalam buffer.
- Mengganti `System.out` dan `System.err` dengan objek `PrintStream` yang terhubung ke `ByteArrayOutputStream`.

3. Metode `appendToTextArea(final String str)`:

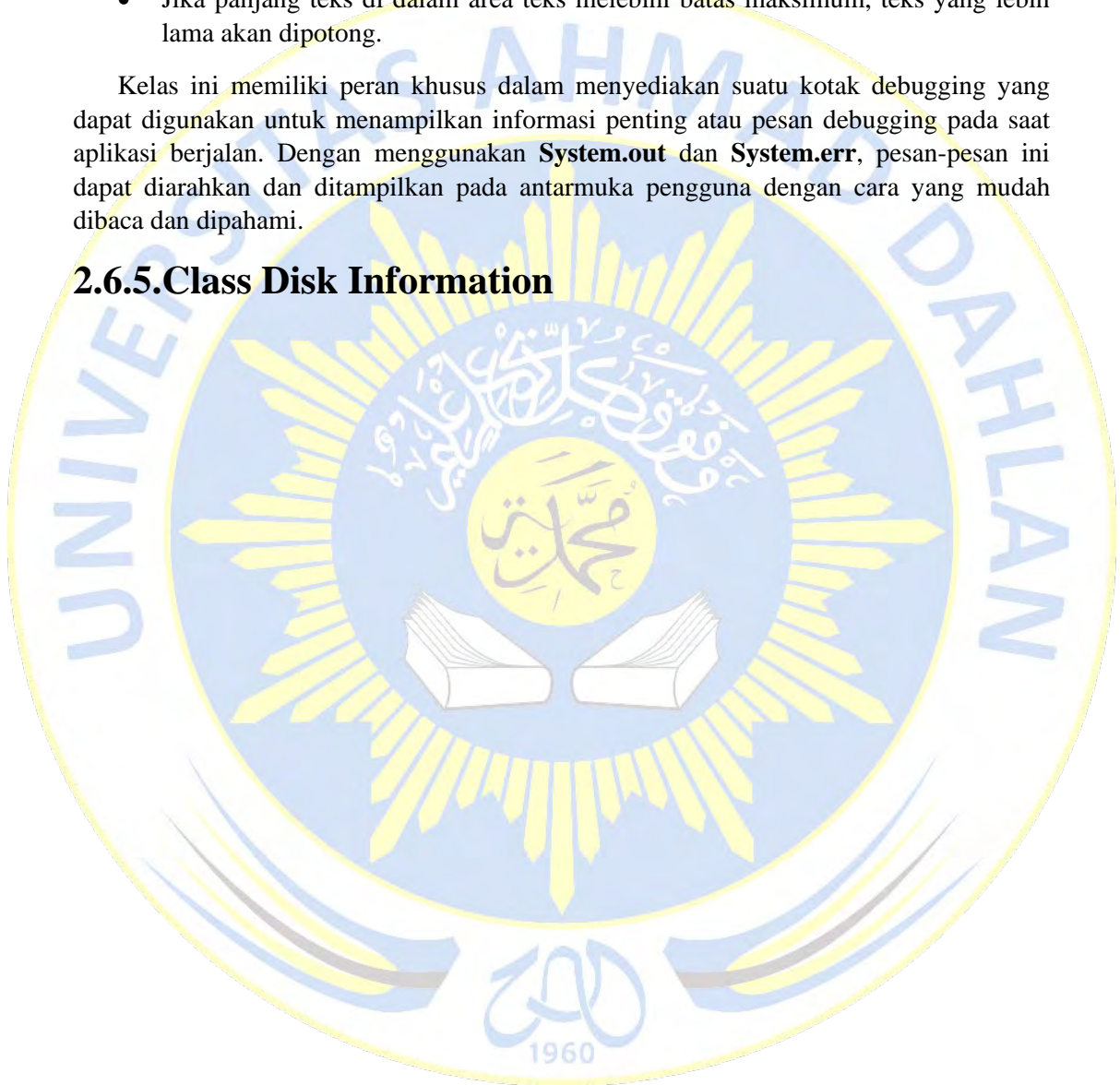
- Metode ini menambahkan string ke area teks (**debugTextArea**) secara asinkron, memastikan operasi UI dilakukan di thread JavaFX.
- Setelah menambahkan teks, area teks di-scroll ke posisi paling bawah agar pengguna dapat melihat pesan terbaru.

4. Metode appendText(String str):

- Metode ini menambahkan string ke area teks (**debugTextArea**), dengan memperhatikan batas maksimum panjang teks.
- Jika panjang teks di dalam area teks melebihi batas maksimum, teks yang lebih lama akan dipotong.

Kelas ini memiliki peran khusus dalam menyediakan suatu kotak debugging yang dapat digunakan untuk menampilkan informasi penting atau pesan debugging pada saat aplikasi berjalan. Dengan menggunakan **System.out** dan **System.err**, pesan-pesan ini dapat diarahkan dan ditampilkan pada antarmuka pengguna dengan cara yang mudah dibaca dan dipahami.

2.6.5. Class Disk Information




```
1 package com.example.compactjv.UI;
2
3 import com.example.compactjv.Controller;
4 import com.example.compactjv.Disk;
5 import javafx.application.Platform;
6 import javafx.concurrent.ScheduledService;
7 import javafx.concurrent.Task;
8 import javafx.geometry.Insets;
9 import javafx.scene.control.Label;
10 import javafx.scene.control.ProgressBar;
11 import javafx.scene.layout.HBox;
12 import javafx.scene.layout.VBox;
13 import javafx.util.Duration;
14
15 public class DiskInformationUI extends Controller {
16
17     public DiskInformationUI() {
18         super();
19         diskContainer = Controller.getInstance().getDiskContainer();
20         setupDiskSizeList();
21         startUpdateService();
22     }
23     private void setupDiskSizeList() {
24         Disk[] disks = Disk.getDisks();
25         diskContainer.setPadding(new Insets(65, 0, 0, 10));
26         for (Disk disk : disks) {
27             VBox diskBox = new VBox();
28             Label diskName = new Label(Character.toString(disk.getLabel()));
29             Label diskFreeSpace = new Label("Free Space: " + disk.getFreeSpace().getSizeFormatted());
30             Label diskTotalSize = new Label("Total Size: " + disk.getTotalSpace().getSizeFormatted());
31             double freeSpaceRatio = (double)disk.getFreeSpace().getSize() / disk.getTotalSpace().getSize();
32             ProgressBar diskSpaceBar = new ProgressBar();
33             diskSpaceBar.setPrefWidth(160);
34             diskSpaceBar.setProgress(1-freeSpaceRatio);
35
36             diskName.setStyle("-fx-font-family: 'Segoe UI Black'; -fx-text-fill: white;");
37             diskFreeSpace.setStyle("-fx-font-family: 'Segoe UI Semilight'; -fx-text-fill: white;");
38             diskTotalSize.setStyle("-fx-font-family: 'Segoe UI Semilight'; -fx-text-fill: white;");
39
40             if(freeSpaceRatio < 0.2) {
41                 diskSpaceBar.setStyle("-fx-accent: red;");
42             } else {
43                 diskSpaceBar.setStyle("-fx-accent: green;");
44             }
45
46             HBox labelAndBar = new HBox();
47             labelAndBar.setSpacing(10);
48             labelAndBar.getChildren().addAll(diskName, diskSpaceBar);
49             diskBox.getChildren().addAll(labelAndBar, diskFreeSpace, diskTotalSize);
50             diskContainer.getChildren().add(diskBox);
51         }
52     }
53
54     private void startUpdateService() {
55         ScheduledService<Void> updateService = new ScheduledService<>() {
56             protected Task<Void> createTask() {
57                 return new Task<>() {
58                     protected Void call() {
59                         Platform.runLater(() -> {
60                             diskContainer.getChildren().clear();
61                             setupDiskSizeList();
62                         });
63                     }
64                 };
65             }
66         };
67     };
68
69     updateService.setPeriod(Duration.seconds(5));
70     updateService.start();
71 }
72 }
```

Gambar 45 class diskInformationUI

Kelas **DiskInformationUI** dalam kode di atas mengelola tampilan informasi disk pada antarmuka pengguna JavaFX. Berikut adalah penjelasan beberapa aspek penting dalam kodenya:

1. Konstruktor DiskInformationUI():

- Konstruktor dipanggil saat menciptakan instans dari kelas ini.
- Memanggil konstruktor kelas induk (**Controller**).
- Mengakses **diskContainer** dari instance **Controller** dan menginisialisasi tampilan daftar informasi disk dengan memanggil metode **setupDiskSizeList()**.
- Memulai layanan penjadwalan (**ScheduledService**) untuk pembaruan periodik tampilan disk menggunakan metode **startUpdateService()**.

2. Metode setupDiskSizeList():

- Metode ini bertanggung jawab untuk mengonfigurasi dan menampilkan informasi disk dalam antarmuka pengguna.
- Mendapatkan array objek **Disk** dengan metode statis **Disk.getDisks()**.
- Menyiapkan tampilan untuk setiap disk, termasuk label nama disk, informasi ruang bebas, informasi ukuran total, dan progress bar untuk memvisualisasikan rasio ruang bebas terhadap ukuran total.
- Styling dilakukan menggunakan CSS dengan menentukan font, warna teks, dan warna aksent (ProgressBar).
- Menambahkan tampilan-tampilan disk ke dalam **diskContainer**, yaitu suatu **VBox** yang berisi informasi-informasi tersebut.

3. Metode startUpdateService():

- Metode ini memulai layanan penjadwalan (**ScheduledService**) untuk melakukan pembaruan periodik tampilan informasi disk.
- Layanan tersebut didefinisikan sebagai sebuah task yang dijalankan oleh **Platform.runLater()** untuk memastikan operasi UI berjalan di thread JavaFX.
- Pembaruan dilakukan setiap 5 detik (sesuai dengan periode yang diatur).
- **ScheduledService dan Task:**
- **ScheduledService** digunakan untuk menjadwalkan tugas tertentu secara periodik.
- Pada implementasi ini, task yang dijadwalkan adalah pembaruan tampilan informasi disk yang diatur oleh metode **setupDiskSizeList()**.

4. Styling dan Tampilan UI:

- Styling diterapkan pada label, ProgressBar, dan container menggunakan CSS.
- Warna ProgressBar diatur berdasarkan rasio ruang bebas terhadap ukuran total, dengan warna merah jika rasio kurang dari 0.2, dan warna hijau jika sebaliknya.

2.6.6. Class File Path UI

```
1 package com.example.compactjv.ui;
2
3 import com.example.compactjv.Controller;
4 import com.example.compactjv.Size;
5 import javafx.application.Platform;
6 import javafx.scene.Node;
7 import javafx.scene.input.DragEvent;
8 import javafx.scene.input.MouseEvent;
9 import javafx.scene.input.TransferMode;
10 import javafx.stage.DirectoryChooser;
11 import javafx.stage.Stage;
12
13
14 public class FilePathUI extends Controller {
15
16     public FilePathUI() {
17         super();
18         setupFilePathField();
19     }
20
21     private void setupFilePathField() {
22         filePathField.textProperty().addListener((observable, oldValue, newValue) -> fieldTextHasChanged());
23         filePathField.setOnMouseClicked(this::handleMouseClickedOnField);
24         filePathField.setOnDragOver(this::handleDragOverField);
25         filePathField.setOnDragDropped(this::handleDragDroppedOnField);
26     }
27
28     private void handleMouseClickedOnField(MouseEvent event) {
29         Stage primaryStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
30         java.io.File selectedDirectory = new DirectoryChooser().showDialog(primaryStage);
31         if (selectedDirectory != null) updateFilePath(selectedDirectory);
32     }
33
34     private void handleDragOverField(DragEvent event) {
35         if (event.getGestureSource() != filePathField && event.getDragboard().hasFiles())
36             event.acceptTransferModes(TransferMode.COPY_OR_MOVE);
37         event.consume();
38     }
39
40     private void handleDragDroppedOnField(DragEvent event) {
41         if (event.getDragboard().hasFiles()) {
42             java.io.File file = event.getDragboard().getFiles().get(0);
43             if (file.isDirectory()) {
44                 updateFilePath(file);
45                 event.setDropCompleted(true);
46             }
47         }
48         event.consume();
49     }
50
51     private void fieldTextHasChanged() {
52         String filePath = compact.getFilePath();
53         compressButton.setDisable(true);
54         decompressButton.setDisable(true);
55         currentSizeLabel.setText("Loading...");
56         sizeOnDiskLabel.setText("Loading...");
57         if (compact.isValidDirectory(filePath)) prepareFolder(filePath);
58     }
59
60     private void updateFilePath(java.io.File file) {
61         compact.setFilePath(file.getAbsolutePath());
62         filePathField.setText(file.getName());
63     }
64
65     private void prepareFolder(String filePath) {
66         executorServiceManager.executeTask(() -> {
67             boolean isCompressed = compact.isCompressed(filePath);
68             Size size = compact.calculateFolderSize(filePath);
69             Platform.runLater(() -> updateUI(isCompressed, size));
70         });
71     }
72
73     private void updateUI(boolean isCompressed, Size size) {
74         compressButton.setDisable(false);
75         decompressButton.setDisable(!isCompressed);
76         compressionAlgorithmChoiceBox.setDisable(false);
77         compressButton.setText(isCompressed ? "Compress Again" : "Compress");
78         currentSizeLabel.setText(size.getSizeFormatted());
79         sizeOnDiskLabel.setText(isCompressed ? size.getSizeOnDiskFormatted() : "???");
80         totalFolderOnFile.setText(Long.toString(compact.getTotalFilesInFolder()));
81     }
82 }
83
84
85
```

Gambar 46 class FilePathUI.

Kelas **FilePathUI** dalam kode di atas bertanggung jawab untuk mengelola interaksi antara pengguna dan elemen-elemen UI yang terkait dengan pemilihan dan tampilan path/file dalam aplikasi. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. Konstruktor FilePathUI():

- Konstruktor dipanggil saat menciptakan instans dari kelas ini.
- Memanggil konstruktor kelas induk (**Controller**).
- Menginisialisasi dan mengatur elemen UI terkait path/file dengan memanggil metode **setupFilePathField()**.

2. Metode setupFilePathField():

- Metode ini menangani konfigurasi elemen UI terkait path/file.
- Menambahkan listener ke properti teks dari **filePathField**, sehingga metode **fieldTextHasChanged()** akan dipanggil ketika terjadi perubahan pada teks.
- Mengatur handler untuk event klik mouse (**setOnMouseClicked**) untuk memproses pemilihan direktori saat pengguna mengklik pada field path/file.
- Mengatur handler untuk event drag over (**setOnDragOver**) dan drag dropped (**setOnDragDropped**) untuk memproses pemilihan direktori saat pengguna melakukan drag and drop ke dalam field path/file.

3. Metode-metode Event Handler:

- **handleMouseClickedOnField(MouseEvent event):** Menangani klik pada field path/file untuk memilih direktori menggunakan **DirectoryChooser**.
- **handleDragOverField(DragEvent event):** Menangani event drag over pada field path/file, memastikan bahwa drag and drop hanya diterima jika ada file yang di-drag dan bukan folder.
- **handleDragDroppedOnField(DragEvent event):** Menangani event drag dropped pada field path/file, memperbarui path/file jika file yang di-drop adalah direktori.

4. Metode fieldTextHasChanged():

- Dipanggil ketika teks pada **filePathField** berubah.
- Mendapatkan path dari **compact.getFilePath()**.
- Mempersiapkan folder jika path merupakan direktori yang valid.

5. Metode updateFilePath(java.io.File file):

- Memperbarui path/file dalam aplikasi berdasarkan file yang dipilih atau di-drop.

6. Metode prepareFolder(String filePath):

- Menyiapkan folder dengan melakukan pengecekan apakah folder sudah di-compress atau belum.
- Menjalankan tugas secara asinkron menggunakan **executorServiceManager**.
- Mengupdate UI setelah penyelesaian tugas.

7. Metode updateUI(boolean isCompressed, Size size):

- Mengupdate elemen-elemen UI sesuai dengan status compress dan ukuran folder.
- Mengatur enable/disable tombol-tombol dan memperbarui label-label dengan informasi yang sesuai.

Kelas ini memungkinkan pengguna untuk memilih folder, baik melalui dialog atau dengan metode drag and drop. Selain itu, kelas ini juga bertanggung jawab untuk

mempersiapkan folder dan mengupdate UI berdasarkan perubahan yang terjadi pada path/file yang dipilih.

2.6.7. Class Information UI



```
1 package com.example.compactjv.UI;
2
3 public interface InformationUI {
4     void setupUsageLabel();
5 }
6
```

Kode di atas mendefinisikan sebuah antarmuka (interface) bernama **InformationUI**. Antarmuka ini memiliki satu metode yaitu **setupUsageLabel()**, yang harus diimplementasikan oleh kelas-kelas yang mengimplementasi antarmuka ini. Dengan adanya antarmuka ini, kelas-kelas yang memerlukan metode **setupUsageLabel()** dapat mengimplementasikan antarmuka ini untuk menyediakan perilaku khusus yang sesuai dengan kebutuhan masing-masing kelas.

Berikut adalah beberapa poin penting terkait antarmuka **InformationUI**:

1. Antarmuka (Interface):

- Antarmuka adalah kontrak tanpa implementasi, yang menyatakan bahwa kelas-kelas yang mengimplementasikan antarmuka ini harus menyediakan implementasi untuk semua metode yang dideklarasikan dalam antarmuka.
- Antarmuka menyediakan suatu cara untuk mendefinisikan perilaku yang harus diimplementasikan oleh kelas-kelas yang menggunakan antarmuka tersebut.

2. Metode **setupUsageLabel()**:

- Antarmuka ini menyediakan satu metode, yaitu **setupUsageLabel()**, yang harus diimplementasikan oleh kelas-kelas yang mengadopsi (**implements**) antarmuka ini.
- Metode ini memberikan fleksibilitas kepada kelas-kelas yang menggunakan antarmuka **InformationUI** untuk menyediakan berbagai macam implementasi yang sesuai dengan kebutuhan spesifik kelas tersebut.

Dengan adanya antarmuka **InformationUI**, suatu kelas dapat menyatakan bahwa ia memiliki kemampuan untuk menyediakan perilaku khusus yang terkait dengan pengaturan label informasi, sehingga meningkatkan struktur dan fleksibilitas dalam desain aplikasi.

2.6.8. Class Loading bar

```
1 package com.example.compactjv.ui;
2
3 import com.example.compactjv.Controller;
4 import com.example.compactjv.ExecutorServiceManager;
5 import javaFX.application.Platform;
6
7 import java.util.concurrent.Future;
8 import java.util.concurrent.ScheduledExecutorService;
9 import java.util.concurrent.TimeUnit;
10
11 public class LoadingBarUI extends Controller {
12
13     public LoadingBarUI(Future<?> future, String filePath, boolean isCompression) {
14         super();
15         monitorTaskUntilCompletion(future, filePath, isCompression, executorServiceManager);
16     }
17
18     public void monitorTaskUntilCompletion(Future<?> future, String filePath, boolean isCompression, ExecutorServiceManager executorManager) {
19         executorManager.submitTask(() -> {
20             while (!future.isDone()) {
21                 compact.getTotalFolderCompressed(filePath);
22                 updatePercentagelabel(isCompression);
23             }
24
25             if (future.isDone()) {
26                 Platform.runLater(() -> {
27                     percentagelabel.setText("Done!");
28                     progressBar.setVisible(false);
29                     ScheduledExecutorService scheduler = executorManager.getScheduler();
30                     scheduler.schedule(() -> Platform.runLater(() -> {
31                         percentagelabel.setText("");
32                     })), 5, TimeUnit.SECONDS);
33                 });
34             }
35         });
36     }
37
38     private void updatePercentagelabel(boolean isCompression) {
39         long totalCompressed = compact.getTotalCompressed();
40         long totalDecompressed = compact.getTotalDecompressed();
41         long total = totalCompressed + totalDecompressed;
42         double percentage = calculatePercentage(totalCompressed, totalDecompressed, total, isCompression);
43         Platform.runLater(() -> {
44             percentagelabel.setText(String.format("%.2f", percentage) + "%");
45             progressBar.setProgress(percentage / 100);
46             progressBar.setVisible(true);
47         });
48     }
49
50     private double calculatePercentage(long totalCompressed, long totalDecompressed, long total, boolean isCompression) {
51         return (double) ((isCompression ? totalCompressed : totalDecompressed) / total * 100);
52     }
53 }
54 }
```

Gambar 47 class LoadingBarUI.

Kelas **LoadingBarUI** dalam kode di atas dirancang untuk memonitor dan menampilkan status kemajuan dari sebuah tugas (task), seperti kompresi atau dekompresi, melalui suatu loading bar pada antarmuka pengguna. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. **Konstruktor LoadingBarUI(Future<?> future, String filePath, boolean isCompression):**
 - Konstruktor menerima tiga parameter: **Future<?> future** untuk memonitor status tugas, **String filePath** untuk path/file yang sedang diolah, dan **boolean isCompression** untuk menentukan apakah tugas tersebut merupakan kompresi atau dekompresi.
 - Memanggil konstruktor kelas induk (**Controller**).
 - Memanggil metode **monitorTaskUntilCompletion()** untuk memulai pemantauan tugas sampai selesai.
2. **Metode monitorTaskUntilCompletion(Future<?> future, String filePath, boolean isCompression, ExecutorServiceManager executorManager):**
 - Metode ini memonitor status tugas yang diwakili oleh **future** dan memperbarui loading bar serta label persentase sesuai dengan kemajuan tugas.

- Dalam suatu thread terpisah yang diatur oleh **ExecutorServiceManager**, metode ini terus-menerus memanggil metode **updatePercentageLabel()** dan menghentikan pemantauan ketika tugas selesai.
- Setelah tugas selesai, UI diperbarui untuk menampilkan pesan "Done!" dan loading bar disembunyikan. Setelah 5 detik, label persentase dihapus.

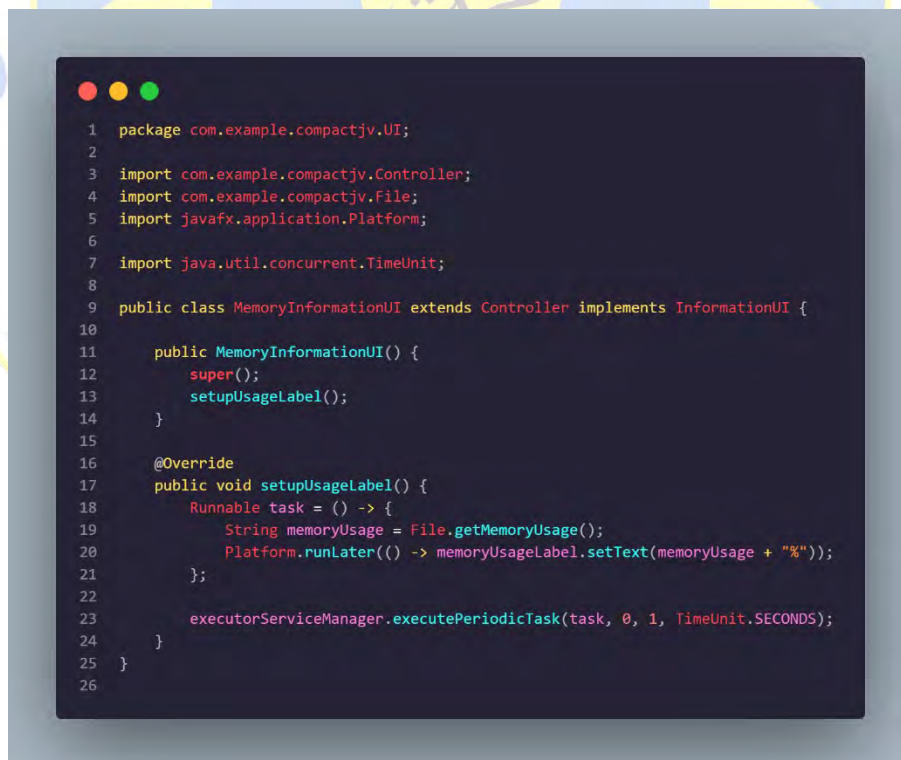
3. Metode updatePercentageLabel(boolean isCompression):

- Metode ini menghitung persentase kemajuan tugas dan memperbarui label persentase serta nilai progress bar di thread UI (JavaFX).
- Menggunakan metode **calculatePercentage()** untuk menghitung persentase berdasarkan total file yang telah diompres atau didekompres.
- Menggunakan **Platform.runLater()** untuk memastikan bahwa pembaruan UI dilakukan di thread JavaFX.

4. Metode calculatePercentage(long totalCompressed, long totalDecompressed, long total, boolean isCompression):

- Metode ini menghitung persentase kemajuan tugas berdasarkan total file yang telah diompres atau didekompres.
- Kelas **LoadingBarUI** dirancang untuk memberikan pengalaman pengguna yang dinamis dan informatif selama tugas kompresi atau dekompresi sedang berlangsung, dengan pemantauan kemajuan yang real-time melalui loading bar dan label persentase.

2.6.9. Class Memory Information



```
1 package com.example.compactjv.UI;
2
3 import com.example.compactjv.Controller;
4 import com.example.compactjv.File;
5 import javafx.application.Platform;
6
7 import java.util.concurrent.TimeUnit;
8
9 public class MemoryInformationUI extends Controller implements InformationUI {
10
11     public MemoryInformationUI() {
12         super();
13         setupUsageLabel();
14     }
15
16     @Override
17     public void setupUsageLabel() {
18         Runnable task = () -> {
19             String memoryUsage = File.getMemoryUsage();
20             Platform.runLater(() -> memoryUsageLabel.setText(memoryUsage + "%"));
21         };
22
23         executorServiceManager.executePeriodicTask(task, 0, 1, TimeUnit.SECONDS);
24     }
25 }
26
```

Gambar 48 class MemoryInformationUI.

Kelas **MemoryInformationUI** dalam kode di atas bertanggung jawab untuk menyajikan informasi penggunaan memori pada antarmuka pengguna dengan memperbarui label penggunaan memori secara periodik. Berikut adalah beberapa poin kunci dalam kode tersebut:

1. Konstruktor MemoryInformationUI():

- Konstruktor dipanggil saat menciptakan instans dari kelas ini.
- Memanggil konstruktor kelas induk (**Controller**).
- Menginisialisasi dan mengatur label penggunaan memori dengan memanggil metode **setupUsageLabel()**.

2. Implementasi Antarmuka InformationUI:

Kelas ini mengimplementasikan antarmuka **InformationUI** dan menyediakan implementasi dari metode **setupUsageLabel()**. Metode ini akan dijalankan secara periodik untuk memperbarui label penggunaan memori.

3. Metode setupUsageLabel():

- Metode ini merupakan implementasi dari antarmuka **InformationUI**.
- Membuat suatu **Runnable task** yang akan dijalankan secara periodik.
- **task** menggunakan metode statis **File.getMemoryUsage()** untuk mendapatkan informasi penggunaan memori.
- Memastikan bahwa pembaruan UI dilakukan di thread JavaFX menggunakan **Platform.runLater()**.
- Menjadwalkan eksekusi **task** secara periodik dengan memanggil **executorServiceManager.executePeriodicTask()** setiap 1 detik.

2.6.10. Class Navbar UI

```
1 package com.example.compactjv.UI;
2
3 import com.example.compactjv.Controller;
4 import javafx.animation.KeyFrame;
5 import javafx.animation.KeyValue;
6 import javafx.animation.Timeline;
7 import javafx.util.Duration;
8 import lombok.Getter;
9 import lombok.Setter;
10
11 @Getter @Setter
12 public class NavbarUI extends Controller {
13     private boolean isNavbarVisible;
14
15     public NavbarUI() {
16         super();
17         hideNavbar();
18         hamburgerButton.setOnMouseClicked(event -> toggleNavbar());
19     }
20
21     private void toggleNavbar() {
22         if (isNavbarVisible) {
23             hideNavbar();
24         } else {
25             showNavbar();
26         }
27     }
28
29     private void showNavbar() {
30         Timeline timeline = createTimeline(200);
31         infoText.visibleProperty().setValue(true);
32         homeText.visibleProperty().setValue(true);
33         timeline.setOnFinished(e -> navbar.getChildren().addAll(infoText, homeText));
34         timeline.play();
35         isNavbarVisible = true;
36     }
37
38     private void hideNavbar() {
39         navbar.getChildren().removeAll(infoText, homeText);
40         Timeline timeline = createTimeline(65);
41         timeline.play();
42         isNavbarVisible = false;
43     }
44
45     /**
46      * Create a timeline for animating the navbar
47      * @param width target width
48      * @return timeline object
49      */
50     private Timeline createTimeline(int width) {
51         KeyValue kv = new KeyValue(navbar.prefWidthProperty(), width);
52         KeyFrame kf = new KeyFrame(Duration.millis(300), kv);
53
54         Timeline timeline = new Timeline();
55         timeline.getKeyFrames().add(kf);
56
57         return timeline;
58     }
59 }
60
```

Gambar 49 class NavbarUI.

Kelas **NavbarUI** dalam kode di atas mengelola tampilan navbar pada antarmuka pengguna dengan kemampuan untuk menyembunyikan dan menampilkan elemen-elemen navbar menggunakan animasi. Berikut adalah beberapa poin kunci dalam kode tersebut:

1. Atribut isNavbarVisible:

- Atribut ini digunakan untuk melacak apakah navbar sedang terlihat atau tidak.

2. Konstruktor NavbarUI():

- Konstruktor dipanggil saat menciptakan instans dari kelas ini.

- Memanggil konstruktor kelas induk (**Controller**).
- Menginisialisasi dan menyembunyikan navbar saat aplikasi dimulai.
- Menetapkan event handler untuk mouse click pada elemen **hamburgerButton**, sehingga memanggil metode **toggleNavbar()** saat tombol hamburger diklik.

3. Metode toggleNavbar():

- Metode ini memeriksa status terlihat atau tidaknya navbar dan memanggil metode **showNavbar()** atau **hideNavbar()** sesuai dengan kondisi.

4. Metode showNavbar():

- Metode ini menggunakan **Timeline** untuk membuat animasi pengungkapan navbar.
- Membuat timeline dengan memanggil metode **createTimeline(200)** untuk mengatur lebar target navbar.
- Mengatur properti **visible** pada elemen-elemen navbar dan menambahkannya kembali ke dalam **navbar** setelah animasi selesai.
- Menetapkan nilai **isNavbarVisible** menjadi **true**.

5. Metode hideNavbar():

- Metode ini menggunakan **Timeline** untuk membuat animasi penyembunyian navbar.
- Menghapus elemen-elemen navbar dari dalam **navbar** saat animasi dimulai.
- Membuat timeline dengan memanggil metode **createTimeline(65)** untuk mengatur lebar target navbar.
- Menetapkan nilai **isNavbarVisible** menjadi **false**.

6. Metode createTimeline(int width):

- Metode ini membuat objek **Timeline** untuk animasi lebar navbar.
- Menggunakan **KeyValue** dan **KeyFrame** untuk menentukan animasi yang diinginkan, dengan durasi 300 milidetik.

2.6.11. Class Tool Tip UI



Gambar 50 class TooltipUI.

Kelas **TooltipUI** dalam kode di atas bertanggung jawab untuk mengatur dan menampilkan tooltip pada berbagai elemen antarmuka pengguna. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. **Konstruktor TooltipUI):**

- Konstruktor dipanggil saat menciptakan instans dari kelas ini.
- Memanggil konstruktor kelas induk (**Controller**).
- Memanggil metode **setupTooltips()** untuk mengatur semua tooltip yang diperlukan.

2. **Metode setupTooltips():**

- Metode ini mengatur tooltip untuk berbagai elemen antarmuka pengguna dengan memanggil metode **createTooltip()** dan **Tooltip.install()** untuk setiap elemen.
- Setiap tooltip berisi deskripsi singkat tentang fungsi dari elemen yang bersangkutan.

3. **Metode createTooltip(String text):**

- Metode ini membuat objek **Tooltip** dengan teks dan penyesuaian tertentu.
- Menetapkan waktu tampilan tooltip dengan **setShowDelay(Duration.seconds(0.5))**.
- Mengatur gaya tooltip dengan menggunakan properti CSS seperti font, ukuran font, warna latar belakang, dan warna teks.
- Mengatur posisi tooltip agar mengikuti pergerakan mouse saat berada di atas elemen yang terkait.

Dengan adanya kelas **TooltipUI**, pengguna mendapatkan informasi tambahan tentang fungsi dari setiap elemen antarmuka pengguna melalui tooltip. Penyesuaian seperti warna, font, dan waktu tampilan tooltip memberikan pengalaman pengguna yang lebih baik dan dapat dipahami dengan mudah.

2.6.12. Class Windows Controller UI

```
1 package com.example.compactjv.ui;
2
3 import com.example.compactjv.Controller;
4 import javafx.animation.ScaleTransition;
5 import javafx.event.ActionEvent;
6 import javafx.event.Event;
7 import javafx.event.EventListener;
8 import javafx.scene.Node;
9 import javafx.scene.control.Button;
10 import javafx.scene.layout.StackPane;
11 import javafx.util.Duration;
12
13 public class WindowControllerUI extends Controller {
14
15     public WindowControllerUI() {
16         super();
17         setButtonProperties(closeButton, event -> System.exit(0));
18         setButtonProperties(minimizeButton, event -> ((Stage) event.getSource()).getScene().getWindow().setIconified(true));
19     }
20
21     private void setButtonProperties(Button button, EventListener<ActionEvent> eventHandler) {
22         button.setOnAction(eventHandler);
23
24         ScaleTransition st = new ScaleTransition(Duration.millis(200), button);
25         st.setByX(0.15);
26         st.setByY(0.15);
27         st.setCycleCount(2);
28         st.setAutoReverse(true);
29
30         button.setOnMouseEntered(event -> st.playFromStart());
31         button.setOnMouseExited(event -> {
32             button.setScaleX(1.0);
33             button.setScaleY(1.0);
34         });
35     }
36 }
37
```

Gambar 51 class WindowsControllerUI.

Kelas **WindowControlsUI** dalam kode di atas bertanggung jawab untuk mengelola tombol kontrol jendela, seperti tombol close dan minimize pada antarmuka pengguna. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. **Konstruktor WindowControlsUI():**
 - Konstruktor dipanggil saat menciptakan instans dari kelas ini.
 - Memanggil konstruktor kelas induk (**Controller**).
 - Mengatur properti dan perilaku tombol-tombol kontrol jendela dengan memanggil metode **setButtonProperties()** untuk tombol close dan minimize.
2. **Metode setButtonProperties(Button button, EventHandler<ActionEvent> eventHandler):**
 - Metode ini mengatur properti dan perilaku untuk tombol yang diberikan.
 - Menetapkan event handler untuk tombol, yang akan dijalankan saat tombol ditekan.
 - Membuat animasi skala menggunakan **ScaleTransition** untuk memberikan efek animasi saat tombol dimasukkan dan dikeluarkan.
 - Mengatur skala tombol saat mouse masuk ke atas tombol, dan mengembalikan ke skala asli saat mouse keluar dari tombol.
3. **Lambda Expression untuk Event Handler:**
 - Menggunakan ekspresi lambda untuk menyederhanakan definisi event handler.
 - Tombol close (**closeButton**) menggunakan ekspresi lambda untuk menutup aplikasi saat tombol ditekan (**System.exit(0)**).
 - Tombol minimize (**minimizeButton**) menggunakan ekspresi lambda untuk mengonfirmasi tombol minimize pada jendela aplikasi.
4. **Animasi Skala (ScaleTransition):**
 - Membuat objek **ScaleTransition** untuk animasi skala pada tombol.
 - Menetapkan durasi animasi, perubahan skala (**setByX** dan **setByY**), jumlah siklus animasi, dan otomatis memutar mundur animasi (**setAutoReverse**).
 - Menetapkan event handler untuk memulai animasi saat mouse masuk ke atas tombol.

Class Application

```
1 package com.example.compactjv;
2 import javafx.fxml.FXMLLoader;
3 import javafx.scene.Scene;
4 import javafx.scene.image.Image;
5 import javafx.scene.paint.Color;
6 import javafx.stage.Stage;
7 import javafx.stage.StageStyle;
8 import java.io.IOException;
9 import java.util.Objects;
10
11 public class Application extends javafx.application.Application {
12     private static final String TITLE = "CompactJV";
13     private static final String ICON_PATH = "com/example/compactjv/app.png";
14     private static final String VIEW_PATH = "View.fxml";
15
16     private double xOffset = 0;
17     private double yOffset = 0;
18
19     @Override
20     public void start(Stage stage) {
21         try {
22             FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(VIEW_PATH));
23             Scene scene = new Scene(fxmlLoader.load());
24
25             stage.setResizable(false);
26             scene.setFill(Color.TRANSPARENT);
27
28             setUpMouseListeners(scene, stage);
29
30             stage.setTitle(TITLE);
31             stage.getIcons().add(new Image(Objects.requireNonNull(getClass().getResourceAsStream(ICON_PATH))));
32             stage.setScene(scene);
33             stage.show();
34         } catch (IOException e) {
35             e.printStackTrace();
36         }
37     }
38
39     private void setUpMouseListeners(Scene scene, Stage stage) {
40         scene.setOnMousePressed(event -> {
41             xOffset = event.getSceneX();
42             yOffset = event.getSceneY();
43         });
44
45         scene.setOnMouseDragged(event -> {
46             stage.setX(event.getScreenX() - xOffset);
47             stage.setY(event.getScreenY() - yOffset);
48         });
49     }
50
51     public static void main(String[] args) {
52         launch();
53     }
54 }
55
```

Kelas ini adalah bagian dari Java yang menggunakan JavaFX untuk mengembangkan aplikasi berbasis Graphical User Interface (GUI). Kelas ini mendefinisikan beberapa variabel, termasuk judul aplikasi CompactJV, path ikon aplikasi, dan path file FXML yang akan digunakan untuk tampilan aplikasi. Kemudian, ada dua variabel, xOffset dan yOffset, yang digunakan untuk mencatat posisi mouse saat jendela aplikasi di-drag.

Metode start() dipanggil ketika aplikasi JavaFX dimulai. Di dalamnya, stage dan scene diatur, file FXML dimuat, dan jendela aplikasi ditampilkan. Metode setUpMouseListeners() menentukan apa yang akan terjadi saat mouse ditekan dan digerakkan di dalam jendela aplikasi. Ini memungkinkan pengguna untuk menggerakkan jendela aplikasi di sekitar layar.

Terakhir, metode main() digunakan untuk memulai aplikasi JavaFX. Secara keseluruhan, kelas ini mendefinisikan bagaimana aplikasi GUI dibuat dan bagaimana interaksi mouse dikelola.

2.6.13. Class Command Runner

```
1 package com.example.compactjv;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6
7 public class CommandRunner {
8     private static final String CMD = "cmd.exe";
9     public static String runCommand(String command, boolean withOutput) {
10         StringBuilder output = new StringBuilder();
11         try {
12             ProcessBuilder processBuilder = new ProcessBuilder(CMD, "/c", command);
13             Process process = processBuilder.start();
14             BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
15             String line;
16             while ((line = reader.readLine()) != null) {
17                 output.append(line).append("\n");
18             }
19             process.waitFor();
20         } catch (IOException | InterruptedException e) {
21             e.printStackTrace();
22         }
23         return withOutput ? output.toString() : null;
24     }
25
26     public static String runCommandWithDebug(String command, boolean withOutput) {
27         StringBuilder output = new StringBuilder();
28         try {
29             ProcessBuilder processBuilder = new ProcessBuilder(CMD, "/c", command);
30             Process process = processBuilder.start();
31             BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
32             String line;
33             while ((line = reader.readLine()) != null) {
34                 output.append(line).append("\n");
35                 System.out.println(line);
36             }
37             process.waitFor();
38         } catch (IOException | InterruptedException e) {
39             e.printStackTrace();
40         }
41         return withOutput ? output.toString() : null;
42     }
43 }
44
```

Gambar 52 class CommandRunner.

Kelas **CommandRunner** dalam kode di atas bertanggung jawab untuk menjalankan perintah di Command Prompt (cmd.exe) dan mengembalikan hasilnya. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. **Variabel Konstan:**

- **CMD:** String yang berisi nama perintah Command Prompt (**cmd.exe**).

2. **Metode runCommand(String command, boolean withOutput):**

- Metode ini menerima perintah yang akan dijalankan dan parameter boolean (**withOutput**) yang menentukan apakah hasil keluaran dari perintah tersebut akan dikembalikan.
- Membuat objek **ProcessBuilder** untuk membuat dan mengonfigurasi proses.
- Membuat objek **Process** dengan menggunakan **start()** dari **ProcessBuilder**.
- Membuat objek **BufferedReader** untuk membaca output dari proses.
- Menunggu proses selesai dengan menggunakan **waitFor()**.

- Jika **withOutput** adalah **true**, mengembalikan hasil keluaran dari perintah; jika tidak, mengembalikan **null**.
- Metode **runCommandWithDebug(String command, boolean withOutput)**:
- Metode ini serupa dengan **runCommand**, namun juga mencetak output perintah ke **System.out** selama eksekusi.
- Berguna untuk tujuan debug dan monitoring eksekusi perintah.

3. Exception Handling:

- Menggunakan blok **try-catch** untuk menangani **IOException** dan **InterruptedException** yang dapat terjadi selama eksekusi perintah.
- Pada kasus exception, hanya mencetak stack trace ke **System.err**.
- Kelas **CommandRunner** ini memberikan fungsionalitas untuk menjalankan perintah di Command Prompt secara asinkron, dan hasil keluarannya dapat diambil melalui **runCommand** atau dicetak selama eksekusi dengan menggunakan **runCommandWithDebug**.

2.6.14. Class Compressor



```
1 package com.example.compactjv;
2
3 public class Compressor {
4     private static final String START_COMMAND = "start /B cmd.exe /c compact";
5     public void compress(String filePath, String algorithm) {
6         CommandRunner.runCommandWithDebug(START_COMMAND + " /A /C /I /S /F /EXE:" + algorithm + " /s:" + filePath + "\"", true);
7     }
8
9
10    public void decompress(String filePath) {
11        CommandRunner.runCommandWithDebug(START_COMMAND + " /U /I /A /F /s:" + filePath + "\"", true);
12    }
13 }
14
```

Gambar 53 class Compressor.

Kelas **Compressor** dalam kode di atas bertanggung jawab untuk melakukan kompresi dan dekompresi file menggunakan perintah **compact** di Command Prompt. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. Variabel Konstan:

- **START_COMMAND**: String yang berisi perintah awal untuk membuka cmd.exe secara asinkron (**start /B cmd.exe /c compact**).

2. Metode **compress(String filePath, String algorithm)**:

- Metode ini menerima path file (**filePath**) dan algoritma kompresi (**algorithm**) sebagai parameter.
- Menggunakan **CommandRunner.runCommandWithDebug** untuk menjalankan perintah kompresi menggunakan perintah **compact** dengan opsi yang sesuai.
- Opsi **/A /C /I /S /F /EXE:** digunakan untuk menentukan parameter kompresi, seperti algoritma (**/EXE:**), dan path file yang akan dikompresi (**/s:""**).

3. Metode **decompress(String filePath)**:

- Metode ini menerima path file (**filePath**) sebagai parameter.

- Menggunakan **CommandRunner.runCommandWithDebug** untuk menjalankan perintah dekompresi menggunakan perintah **compact** dengan opsi yang sesuai.
- Opsi **/U /I /A /F /s:'''** digunakan untuk menentukan parameter dekompresi dan path file yang akan didekompresi (**/s:'''**).

4. Debugging:

- Menggunakan **CommandRunner.runCommandWithDebug** sehingga output dari eksekusi perintah akan dicetak ke **System.out** selama proses berlangsung. Berguna untuk tujuan debugging dan pemantauan.

2.6.15. Class Controller

```
1 package com.example.compactjv;
2
3 import com.example.compactjv.UI.*;
4 import javafx.fxml.FXML;
5 import javafx.scene.control.*;
6 import javafx.scene.layout.AnchorPane;
7 import javafx.scene.layout.VBox;
8 import lombok.Getter;
9 import lombok.Setter;
10
11 @Setter
12 @Getter
13 public class Controller {
14     @Getter
15     private static Controller instance;
16     protected final ExecutorServiceManager executorServiceManager = new ExecutorServiceManager();
17     @FXML
18     protected Button closeButton, minimizeButton, compressButton, decompressButton, informationButton;
19     @FXML
20     protected TextField filePathField;
21     @FXML
22     protected Label currentSizeLabel, sizeOnDiskLabel, percentageLabel, cpuUsageLabel, memoryUsageLabel, totalFolderOnFile;
23     @FXML
24     protected ChoiceBox<String> compressionAlgorithmChoiceBox;
25     @FXML
26     protected ProgressBar progressBar;
27     @FXML
28     protected Label infoText, homeText;
29     @FXML
30     protected AnchorPane navbar;
31     @FXML
32     protected Button hamburgerButton;
33     @FXML
34     protected VBox diskContainer;
35     @FXML
36     protected TextArea debugTextArea;
37     protected File compact = new File();
38
39     public Controller() {
40         if (instance == null) {
41             instance = this;
42         }
43         compact = Controller.getInstance().getCompact();
44         filePathField = Controller.getInstance().getFilePathField();
45         currentSizeLabel = Controller.getInstance().getCurrentSizeLabel();
46         sizeOnDiskLabel = Controller.getInstance().getSizeOnDiskLabel();
47         totalFolderOnFile = Controller.getInstance().getTotalFolderOnFile();
48         compressButton = Controller.getInstance().getCompressButton();
49         decompressButton = Controller.getInstance().getDecompressButton();
50         percentageLabel = Controller.getInstance().getPercentageLabel();
51         progressBar = Controller.getInstance().getProgressBar();
52         cpuUsageLabel = Controller.getInstance().getCpuUsageLabel();
53         memoryUsageLabel = Controller.getInstance().getMemoryUsageLabel();
54         compressionAlgorithmChoiceBox = Controller.getInstance().getCompressionAlgorithmChoiceBox();
55         infoText = Controller.getInstance().getInfoText();
56         homeText = Controller.getInstance().getHomeText();
57         navbar = Controller.getInstance().getNavbar();
58         hamburgerButton = Controller.getInstance().getHamburgerButton();
59         diskContainer = Controller.getInstance().getDiskContainer();
60         debugTextArea = Controller.getInstance().getDebugTextArea();
61         closeButton = Controller.getInstance().getCloseButton();
62         minimizeButton = Controller.getInstance().getMinimizeButton();
63         informationButton = Controller.getInstance().getInformationButton();
64     }
65
66     public void initialize() {
67         compressionAlgorithmChoiceBox.setDisable(true);
68         percentageLabel.setText("");
69         progressBar.setVisible(false);
70         setupUI();
71     }
72
73     private void setupUI() {
74         new DebuggingBoxUI();
75         new ButtonHandlerUI();
76         new NavbarUI();
77         new WindowControlsUI();
78         new ButtonUI();
79         new FilePathUI();
80         new DiskInformationUI();
81         new CPUInformationUI();
82         new MemoryInformationUI();
83         new ToolTipUI();
84     }
85 }
86
87
```

Kelas **Controller** dalam kode di atas bertindak sebagai pengontrol umum untuk antarmuka pengguna (UI) aplikasi. Berikut adalah penjelasan beberapa poin kunci dalam kode tersebut:

1. **FXML Annotations:**

Menandai atribut dan metode dengan anotasi **@FXML** untuk menandakan bahwa elemen-elemen tersebut dideklarasikan di dalam file FXML dan akan diinjeksikan oleh JavaFX.

2. **Atribut UI:**

- Mendeklarasikan atribut untuk berbagai elemen UI yang dibutuhkan, seperti tombol-tombol, teks field, label, choice box, progress bar, dan area teks.

3. **Instance Singleton:**

- Menerapkan pola desain Singleton dengan menggunakan atribut statis **instance** untuk memastikan bahwa hanya ada satu instance dari kelas **Controller**.

4. **Injeksi FXML:**

- Menggunakan metode **initialize()** yang diannotasi dengan **@FXML** sebagai metode inisialisasi. Metode ini dipanggil setelah seluruh elemen FXML diinjeksikan, dan digunakan untuk mengatur kondisi awal dan UI.

5. **ExecutorServiceManager:**

- Membuat instance dari **ExecutorServiceManager** untuk mengelola eksekusi tugas-tugas secara asinkron.

6. **Metode setupUI():**

- Membuat instance dari berbagai kelas UI seperti **DebuggingBoxUI**, **ButtonHandler**, **NavbarUI**, **WindowControlsUI**, **ButtonUI**, **FilePathUI**, **DiskInformationUI**, **CPUInformationUI**, **MemoryInformationUI**, dan **TooltipUI**.
- Mengatur antarmuka pengguna dan menetapkan perilaku masing-masing elemen UI.

7. **Inisialisasi UI:**

- Di dalam metode **initialize()**, beberapa atribut UI diatur untuk kondisi awal, seperti menonaktifkan **compressionAlgorithmChoiceBox**, mengatur teks label, dan mengatur sifat visible dan progress bar.

8. **Konstruktor dan Inisialisasi:**

- Konstruktor digunakan untuk menginisialisasi atribut **compact** dan elemen UI lainnya.
- Setelah konstruksi, metode **initialize()** dipanggil untuk menyelesaikan proses inisialisasi.

Kelas **Controller** ini berperan sebagai penghubung antara logika aplikasi dan antarmuka pengguna, dengan mengatur perilaku elemen-elemen UI dan mengelola eksekusi tugas-tugas secara asinkron menggunakan **ExecutorServiceManager**.

2.6.16. Class Disk


```
1 package com.example.compactjv;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Getter;
5 import lombok.Setter;
6
7 import java.util.ArrayList;
8 import java.util.Arrays;
9 import java.util.Objects;
10
11 @Getter
12 @Setter
13 @AllArgsConstructor
14 public class Disk {
15     private Size totalSpace;
16     private Size freeSpace;
17     private char label;
18
19     public static Disk[] getDisks() {
20         String output = CommandRunner.runCommand("wmic logicaldisk get name,size,freespace", true);
21         String[] lines = Objects.requireNonNull(output).split("\n");
22
23         ArrayList<Disk> diskList = new ArrayList<>();
24         for (int i = 1; i < lines.length; i++) {
25             String[] parts = lines[i].trim().split("\\s+");
26             if (parts.length == 3) {
27                 try {
28                     long freeSpace = Long.parseLong(parts[0]);
29                     char label = parts[1].charAt(0);
30                     long totalSpace = Long.parseLong(parts[2]);
31
32                     Size freeSpaceSize = new Size(freeSpace, freeSpace);
33                     Size totalSpaceSize = new Size(totalSpace, totalSpace);
34
35                     diskList.add(new Disk(totalSpaceSize, freeSpaceSize, label));
36                 } catch (NumberFormatException e) {
37                     System.out.println("Unable to parse disk information: " + Arrays.toString(parts));
38                 }
39             }
40         }
41         Disk[] disks = new Disk[diskList.size()];
42         diskList.toArray(disks);
43         return disks;
44     }
45 }
46
47 }
```

Gambar 54 class Disk

Kelas **Disk** dalam kode di atas merupakan representasi dari informasi disk pada sistem. Berikut adalah penjelasan beberapa poin kunci dalam kelas **Disk**:

1. **Anotasi Lombok:**

- Menggunakan anotasi Lombok seperti **@Getter**, **@Setter**, dan **@AllArgsConstructor** untuk otomatis menghasilkan getter, setter, dan konstruktor dengan semua parameter.

2. **Atribut Disk:**

- Mendeklarasikan atribut **totalSpace** (ukuran total disk), **freeSpace** (ukuran ruang kosong di disk), dan **label** (label disk).

3. **Metode getDisks():**

- Menggunakan kelas **CommandRunner** untuk menjalankan perintah CMD (**wmic logicaldisk get name,size,freespace**) dan mendapatkan informasi disk.
- Memproses keluaran perintah untuk mendapatkan informasi disk.
- Membuat objek **Disk** untuk setiap baris keluaran yang sesuai dengan format yang diharapkan.
- Menangani pengecualian jika parsing informasi disk tidak berhasil.

4. Array Disk:

- Menggunakan **ArrayList** untuk menyimpan objek-objek **Disk** yang dibuat dari informasi disk.
- Mengonversi **ArrayList** menjadi array **Disk** dan mengembalikannya.

5. Format Informasi Disk:

- Menangani informasi disk yang diperoleh dari perintah CMD.
- Mengonversi ukuran disk dari string menjadi objek **Size**.
- Menghasilkan array objek **Disk** yang berisi informasi disk.

2.6.17. Class Executor Service Manager

```
1 package com.example.compactjv;
2
3 import lombok.Getter;
4
5 import java.util.concurrent.*;
6 @Getter
7 public class ExecutorServiceManager {
8
9     private final ExecutorService executorService;
10    private final ScheduledExecutorService scheduler;
11
12    public ExecutorServiceManager() {
13        this.executorService = Executors.newFixedThreadPool(4);
14        this.scheduler = Executors.newScheduledThreadPool(1);
15    }
16
17    public void executeTask(Runnable task) {
18        executorService.execute(() -> {
19            try {
20                task.run();
21            } catch (Exception e) {
22                e.printStackTrace();
23            }
24        });
25    }
26
27    public ScheduledFuture<?> executePeriodicTask(Runnable task, long initialDelay, long period, TimeUnit unit) {
28        return scheduler.scheduleAtFixedRate(task, initialDelay, period, unit);
29    }
30
31    public Future<?> submitTask(Runnable task) {
32        return executorService.submit(() -> {
33            try {
34                task.run();
35            } catch (Exception e) {
36                e.printStackTrace();
37            }
38        });
39    }
40
41    public void shutdownScheduler() {
42        if (scheduler != null && !scheduler.isShutdown()) {
43            scheduler.shutdown();
44        }
45    }
46 }
47
```

Gambar 55 class ExecutorServiceManager

Kelas **ExecutorServiceManager** bertanggung jawab untuk mengelola **ExecutorService** dan **ScheduledExecutorService** dalam konteks aplikasi. Berikut adalah penjelasan beberapa poin penting dalam kelas ini:

1. Executor Services:

- Mendeklarasikan atribut **executorService** dan **scheduler** sebagai **ExecutorService** dan **ScheduledExecutorService** untuk menangani eksekusi tugas.

2. Konstruktor:

- Dalam konstruktor, menginisialisasi **executorService** dengan **Executors.newFixedThreadPool(4)** untuk membuat pool tetap dengan empat thread dan **scheduler** dengan **Executors.newScheduledThreadPool(1)** untuk membuat scheduler dengan satu thread.

3. Metode executeTask:

- Menerima objek **Runnable** (tugas yang akan dieksekusi) sebagai parameter.
- Eksekusi tugas dalam thread dari **executorService**.
- Menangkap dan mencetak stack trace jika terjadi pengecualian selama eksekusi tugas.

4. Metode executePeriodicTask:

- Menerima objek **Runnable** (tugas periodik yang akan dieksekusi), serta **initialDelay**, **period**, dan **unit** sebagai parameter.
- Menggunakan **scheduler.scheduleAtFixedRate** untuk menjadwalkan eksekusi tugas secara periodik.
- Mengembalikan objek **ScheduledFuture** untuk memungkinkan kontrol lebih lanjut atas tugas yang dijadwalkan.

5. Metode submitTask:

- Menerima objek **Runnable** (tugas yang akan dieksekusi) sebagai parameter.
- Menggunakan **executorService.submit** untuk mengirimkan tugas ke eksekutor.
- Menangkap dan mencetak stack trace jika terjadi pengecualian selama eksekusi tugas.

6. Metode shutdownScheduler:

- Memeriksa apakah **scheduler** tidak null dan belum dimatikan sebelum mencoba untuk mematikan scheduler.
- Menggunakan **scheduler.shutdown** untuk memberhentikan scheduler.

2.6.18. Class File

```
1 package com.example.compactjv;
2
3 import lombok.Getter;
4 import lombok.Setter;
5
6 import java.io.IOException;
7 import java.nio.file.Files;
8 import java.nio.file.Paths;
9 import java.nio.file.Path;
10 import java.util.Arrays;
11 import java.util.Objects;
12
13 @Getter
14 @Setter
15 public class FileV {
16     private static final String CMD = "cmd.exe";
17     private static final String QUERY_CMD = "compact /Q /X /I /S!";
18     private String filePath;
19     private long totalCompressed;
20     private long totalDecompressed;
21     private final Compressor compression = new Compressor();
22     private final int indexOfCompressed = 0;
23     private final int indexDecompressed = 1;
24
25     public void compress(String filePath, String algorithm) {
26         compressor.compress(filePath, algorithm);
27     }
28
29     public void decompress(String filePath) {
30         compressor.decompress(filePath);
31     }
32
33     public static String getCPUsage() {
34         String output = CommandRunner.runCommand("wmic cpu get loadpercentage", true);
35         String[] lines = Objects.requireNonNull(output).split("\n");
36         if (lines.length < 2) {
37             return "0";
38         }
39         String[] parts = lines[2].split("\s+");
40         if (parts.length == 0 || parts[0].isEmpty()) {
41             return "0";
42         }
43         return parts[0];
44     }
45
46     public static String getFreeMemory() {
47         String output = CommandRunner.runCommand("wmic os get freePhysicalMemory, totalVisibleMemorySize /value", true);
48         String[] lines = Objects.requireNonNull(output).split("\n");
49         long[] memory = new long[2];
50         for (String line : lines) {
51             if (line.startsWith("FreePhysicalMemory")) {
52                 String value = line.split("=")[2];
53                 memory[0] = Long.parseLong(value);
54             } else if (line.startsWith("TotalVisibleMemorySize")) {
55                 String value = line.split("=")[2];
56                 memory[1] = Long.parseLong(value);
57             }
58         }
59         double percentage = 100 * (1 - (double)memory[0] / (double)memory[1]);
60         return String.format("%.2f", percentage);
61     }
62
63     public long getTotalFilesInFolder() {
64         String filePath = getFilePath();
65         Path path = Paths.get(filePath);
66         long totalFiles = 0;
67
68         try {
69             totalFiles = Files.walk(path)
70                 .parallel()
71                 .filter(p -> p.toFile().isFile())
72                 .count();
73         } catch (IOException e) {
74             e.printStackTrace();
75         }
76         return totalFiles;
77     }
78
79     public boolean isCompressed(String filePath) {
80         String output = CommandRunner.runCommand(QUERY_CMD + " " + filePath + " ", true);
81         return Arrays.stream(Objects.requireNonNull(output).split("\n"))
82             .filter(line -> line.contains(" are compressed"))
83             .anyMatch(line -> Long.parseLong(line.split("=")[0].replace(" ", ""))
84                 .anyMatch(total -> total != 0));
85     }
86
87     public Size calculateFolderSize(String filePath) {
88         String output = CommandRunner.runCommand(QUERY_CMD + " " + filePath + " ", true);
89         return getSizeFromOutput(Objects.requireNonNull(output));
90     }
91
92     private Size getSizeFromOutput(String output) {
93         long size = 0;
94         long sizeOnDisk = 0;
95         String[] lines = output.split("\n");
96         for (String line : lines) {
97             if (line.contains("total bytes of data")) {
98                 String[] parts = line.split(" ");
99                 size = Long.parseLong(parts[0].replace(" ", "").replace(",", ""));
100                 sizeOnDisk = Long.parseLong(parts[1].replace(" ", "").replace(",", ""));
101                 break;
102             }
103         }
104         return new Size(size, sizeOnDisk);
105     }
106
107     public void getTotalFolderCompressed(String filePath) {
108         String output = CommandRunner.runCommand("compact /Q /X /I /S! " + filePath + " ", true);
109         Arrays.stream(Objects.requireNonNull(output).split("\n"))
110             .filter(line -> line.contains(" are compressed"))
111             .findFirst()
112             .ifPresent(line -> {
113                 String[] parts = line.split(" ");
114                 setTotalCompressed(Long.parseLong(parts[indexOfCompressed].replace(" ", "").replace(",", "")));
115                 setTotalDecompressed(Long.parseLong(parts[indexDecompressed].replace(" ", "").replace(",", "")));
116             });
117     }
118
119     public boolean isValidDirectory(String path) {
120         return path != null && !path.isEmpty() && new java.io.File(path).isDirectory();
121     }
122
123 }
124
125 }
```

Gambar 56 class File.

Kelas **File** bertanggung jawab untuk mengelola operasi berkas, kompresi, dekompresi, dan pengambilan informasi tentang berkas. Berikut adalah beberapa poin utama dalam kelas ini:

1. **Atribut Kelas:**

- Mendeklarasikan beberapa atribut termasuk **filePath** untuk menyimpan jalur berkas, **totalCompressed** dan **totalDecompressed** untuk melacak statistik kompresi/dekompresi, serta objek **Compressor** untuk melakukan operasi kompresi/dekompresi.

2. **Operasi Kompresi dan Dekompresi:**

- Menggunakan objek **Compressor** untuk melakukan operasi kompresi dan dekompresi sesuai dengan jalur berkas dan algoritma yang diberikan.

3. **Informasi Penggunaan CPU dan Memori:**

- Menggunakan kelas **CommandRunner** untuk menjalankan perintah pada sistem operasi dan mendapatkan informasi penggunaan CPU dan memori.

4. **Penghitungan Total Berkas dalam Folder:**

- Menggunakan **Files.walk** untuk menghitung total berkas dalam suatu folder.

5. **Pengecekan Status Kompresi:**

- Menggunakan perintah **compact** untuk mengukur status kompresi dari suatu berkas atau folder.

6. **Perhitungan Ukuran Folder:**

- Menggunakan perintah **compact** untuk mengukur ukuran dan ukuran disk dari suatu folder.

7. **Pemantauan Total Folder Terkompresi:**

- Menggunakan perintah **compact** untuk memantau total folder yang terkompresi, serta mengatur **totalCompressed** dan **totalDecompressed** sesuai hasil perintah.

8. **Pengecekan Kevalidan Direktori:**

- Memeriksa apakah suatu path adalah direktori yang valid.

Kelas **File** ini menyediakan antarmuka untuk berinteraksi dengan berkas, melakukan operasi kompresi/dekompresi, serta mengambil informasi terkait ukuran dan status berkas.

2.6.19. Class Size

```
1 package com.example.compactjv;
2
3 import lombok.Getter;
4 import lombok.RequiredArgsConstructor;
5
6 @Getter
7 @RequiredArgsConstructor
8 public class Size {
9     private static final int SIZE_KB = 1024;
10    private static final int SIZE_MB = SIZE_KB * 1024;
11    private static final int SIZE_GB = SIZE_MB * 1024;
12
13    private final long size;
14    private final long sizeOnDisk;
15
16    public String getSizeFormatted() {
17        return formatSize(size);
18    }
19
20    public String getSizeOnDiskFormatted() {
21        return formatSize(sizeOnDisk);
22    }
23
24    private String formatSize(long size) {
25        if (size < SIZE_KB) {
26            return size + " bytes";
27        } else if (size < SIZE_MB) {
28            return String.format("%.2f KB", (double) size / SIZE_KB);
29        } else if (size < SIZE_GB) {
30            return String.format("%.2f MB", (double) size / SIZE_MB);
31        } else {
32            return String.format("%.2f GB", (double) size / SIZE_GB);
33        }
34    }
35 }
36
```

Kelas **Size** bertanggung jawab untuk merepresentasikan dan memformat ukuran berkas serta ukuran di disk. Berikut adalah beberapa poin utama dalam kelas ini:

1. Atribut Kelas:

- Mendeklarasikan beberapa konstanta yang merepresentasikan ukuran dalam byte, kilobyte (KB), megabyte (MB), dan gigabyte (GB).
- Memiliki dua atribut **size** dan **sizeOnDisk** yang mewakili ukuran berkas dan ukuran di disk.

2. Konstruktor:

- Menggunakan anotasi **@RequiredArgsConstructor** untuk menghasilkan konstruktor berdasarkan atribut yang diberikan saat inisialisasi objek.

3. Metode Pengambilan dan Format Ukuran:


- Menyediakan metode **getSizeFormatted** dan **getSizeOnDiskFormatted** untuk mengambil dan memformat ukuran dan ukuran di disk.
- Menggunakan metode **formatSize** untuk melakukan pemformatan ukuran berdasarkan rentang (bytes, KB, MB, GB).

4. Metode Pembantu Pemformatan:

- Menggunakan metode **formatSize** untuk memformat ukuran berkas dan ukuran di disk dengan presisi desimal tertentu.

Kelas **Size** ini memberikan kemudahan dalam merepresentasikan dan memformat ukuran berkas dan ukuran di disk dalam berbagai satuan, sehingga mudah dipahami oleh pengguna atau sistem.

2.6.20. Class Module `com.example.compactJV`



```
1  module com.example.compactjv {
2      requires javafx.controls;
3      requires javafx.fxml;
4      requires javafx.web;
5
6      requires org.controlsfx.controls;
7      requires net.synedra.validatorfx;
8      requires org.kordamp.ikonli.javafx;
9      requires org.kordamp.bootstrapfx.core;
10     // requires eu.hansolo.tilesfx;
11     requires static lombok;
12     requires java.desktop;
13     opens com.example.compactjv to javafx.fxml;
14     exports com.example.compactjv;
15     exports com.example.compactjv.UI;
16     opens com.example.compactjv.UI to javafx.fxml;
17 }
```

Gambar 57 class module `com.example.compactJV`

Modul `com.example.compactjv` mendeklarasikan dependensi yang diperlukan dan mengatur akses terbuka dan pengiriman dari paket-paket di dalamnya. Berikut adalah beberapa poin kunci dalam deskripsi modul:

1. **Dependensi:**

- **JavaFX Modules:** Membutuhkan modul JavaFX seperti `javafx.controls`, `javafx.fxml`, dan `javafx.web`.
- **External Libraries:** Memerlukan beberapa pustaka eksternal, seperti `org.controlsfx.controls`, `net.synedra.validatorfx`, `org.kordamp.ikonli.javafx`, dan `org.kordamp.bootstrapfx.core`.

2. **Lombok:**

- Menggunakan Lombok (`lombok`) sebagai dependensi statis untuk menghasilkan kode boilerplate secara otomatis.

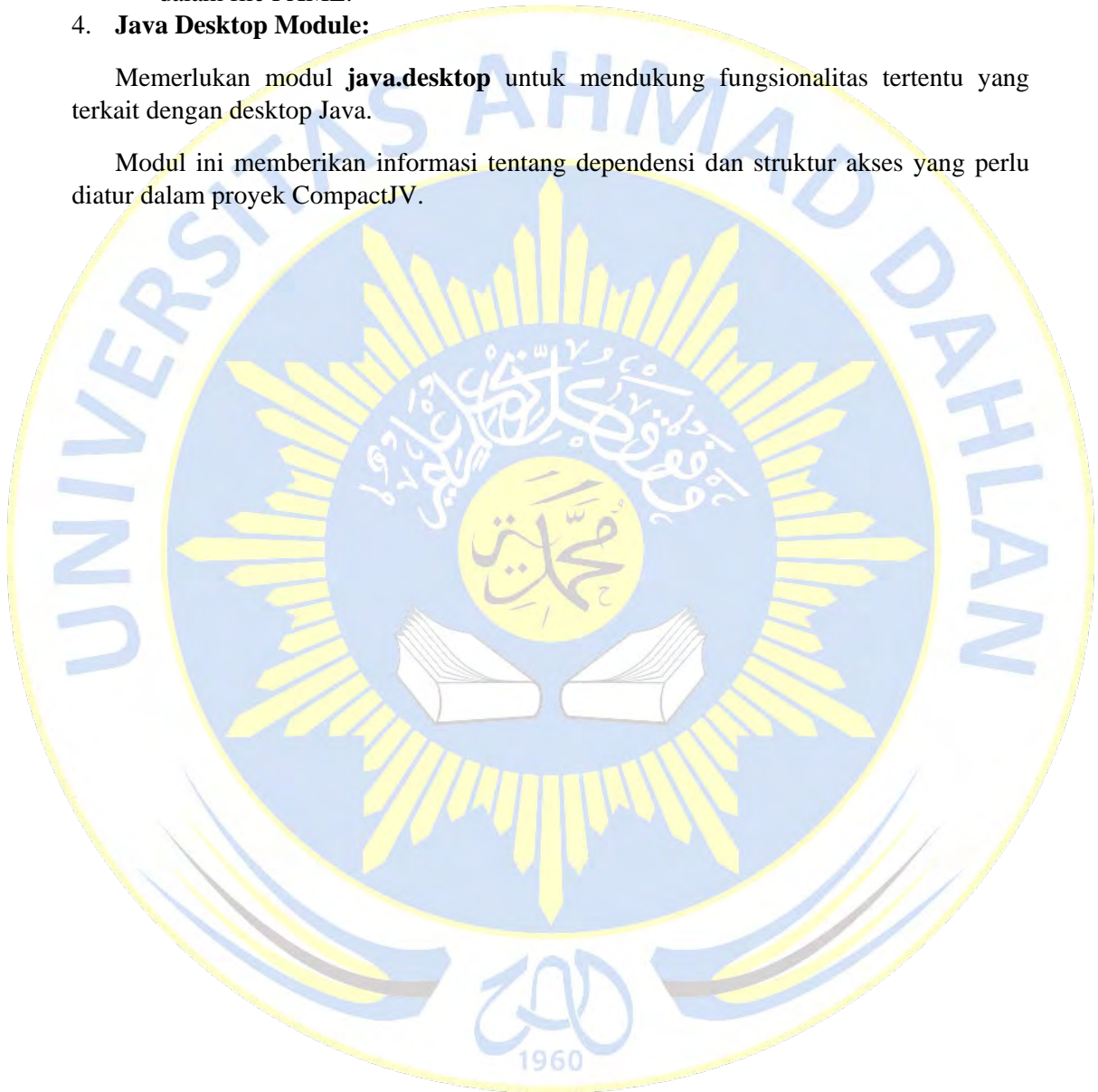
3. **Exports dan Opens:**

- **Exports com.example.compactjv:** Mengekspor paket utama **com.example.compactjv** agar dapat diakses dari luar modul.
- **Exports com.example.compactjv.UI:** Mengekspor paket **com.example.compactjv.UI** agar dapat diakses dari luar modul.
- **Opens com.example.compactjv dan com.example.compactjv.UI to javafx.fxml:** Membuka paket-paket ini ke **javafx.fxml** sehingga dapat digunakan dalam file FXML.

4. Java Desktop Module:

Memerlukan modul **java.desktop** untuk mendukung fungsionalitas tertentu yang terkait dengan desktop Java.

Modul ini memberikan informasi tentang dependensi dan struktur akses yang perlu diatur dalam proyek CompactJV.



BAB III

PENUTUP

3.1. Kesimpulan

Proyek pengembangan CompactJV telah berhasil mencapai tujuan utamanya, yaitu mengembangkan alat kompresi file yang efisien dan mudah digunakan, khususnya untuk software dan game. Dengan memanfaatkan Compact.exe dari Windows SDK, CompactJV mampu mengoptimalkan proses kompresi file dengan menyediakan pilihan algoritma kompresi yang dapat disesuaikan sesuai kebutuhan pengguna. Antarmuka pengguna yang dirancang secara intuitif memungkinkan pengguna untuk melakukan kompresi dan dekompresi file dengan cepat dan mudah.

Selama proses pengembangan, tim telah mendapatkan pengetahuan dan pengalaman berharga dalam pemrograman berorientasi objek, khususnya dalam penggunaan Java dan JavaFX. Selain itu, penerapan pola MVC (Model-View-Controller) telah terbukti efektif dalam memisahkan logika bisnis dari antarmuka pengguna, sehingga memudahkan proses pengembangan dan pemeliharaan kode.

Secara keseluruhan, CompactJV telah berhasil menciptakan solusi kompresi file yang tidak hanya memenuhi kebutuhan pengguna tetapi juga meningkatkan efisiensi penggunaan ruang penyimpanan dan waktu transfer file.

3.2. Kritik dan Saran

Meskipun CompactJV telah berhasil dikembangkan dan diimplementasikan dengan baik, masih terdapat ruang untuk perbaikan. Kritik dan saran yang konstruktif sangat diharapkan untuk membantu meningkatkan kualitas aplikasi di masa mendatang. Beberapa poin yang dapat menjadi perhatian adalah:

- Peningkatan kinerja algoritma kompresi, terutama untuk file dengan ukuran yang sangat besar.
- Pengujian lebih lanjut pada berbagai platform untuk memastikan kompatibilitas dan stabilitas aplikasi.
- Penambahan fitur-fitur baru yang dapat meningkatkan fungsionalitas dan kemudahan penggunaan, seperti integrasi dengan layanan cloud storage.
- Optimasi antarmuka pengguna untuk mendukung resolusi layar yang beragam dan penggunaan pada perangkat dengan layar sentuh.
- Peningkatan dokumentasi dan panduan pengguna untuk memudahkan pengguna baru memahami cara kerja aplikasi.

Dengan menerima kritik dan saran dari berbagai pihak, tim pengembang CompactJV berkomitmen untuk terus belajar dan berkembang dalam menciptakan solusi teknologi yang inovatif dan bermanfaat bagi masyarakat luas.

DAFTAR GAMBAR

Gambar 1 Ilustrasi kepenuhan storage.....	5
Gambar 2 Aplikasi di distribusikan di Github	7
Gambar 3 Flowchart module module-info.java	9
Gambar 4 Flowchart Class Application.java	12
Gambar 5 Flowchart Class CommandRunner.java.....	13
Gambar 6 Flowchart class Compressor.java.....	14
Gambar 7 Flowchart class Disk.java.....	16
Gambar 8 Flowchart class ExecutorServiceManager.java.....	17
Gambar 9 Flowchart class Size.java	18
Gambar 10 Flowchart class File.java.....	20
Gambar 11 Flowchart class Controller.java.....	22
Gambar 12 Flowchart class WindowsConrolsUI.java	24
Gambar 13 Flowchart class TooltipUI.....	26
Gambar 14 Flowchart class NavbarUI.java	28
Gambar 15 Flowchart class MemoryInformationUI.java	30
Gambar 16 Flowchart class LoadingBarUI.java.....	32
Gambar 17 Flowchart class InformationUI.java.....	33
Gambar 18 Flowchart class FilePath.java.....	36
Gambar 19 Flowchart class DiskInformationUI.java	38
Gambar 20 Flowchart class DebuggingBoxUI.java.....	39
Gambar 21 Flowchart Class CPUInformationUI.java	40
Gambar 22 Flowchart class buttonHandlerUI.java.....	43
Gambar 23 Use Case CompactJV	44
Gambar 24 UML Class Diagram, CompactJV	47
Gambar 25 class Application.java	48
Gambar 26 Class Controller.....	49
Gambar 27 Class ExecutorServiceManager	50
Gambar 28 Class File.....	51
Gambar 29 Class Compressor.....	52
Gambar 30 Class Size.	53
Gambar 31 Class ButtonHandler.	54
Gambar 32 Class LoadingBarUI.....	54
Gambar 33 Class FilePathUI.	55
Gambar 34 Interface InformationUI.	56
Gambar 35 Class CPUInformationUI dan MemoryInformationUI.	56
Gambar 36 Tampilan scene builder	58
Gambar 37 Desain CompactJV pada versi Alpha v0.1 sebagai prototype.....	58
Gambar 38 Mencoba berpindah dari prototype menjadi desain modern vertical app desktop.	60
Gambar 39 Modern GUI CompactJV berganti dari vertical menjadi horizontal.	61
Gambar 40 Final design CompactJV	62
Gambar 41 Class Button Handler	65
Gambar 42 class button UI.	67
Gambar 43 class CPUInformationUI.....	69
Gambar 44 class DebuggingBox.....	70

Gambar 45 class diskInformationUI	72
Gambar 46 class FilePathUI.	74
Gambar 47 class LoadingBarUI.....	77
Gambar 48 class MemoryInformationUI.....	78
Gambar 49 class NavbarUI.....	80
Gambar 50 class TooltipUI.....	82
Gambar 51 class WindowsControllerUI.....	83
Gambar 52 class CommandRunner.....	86
Gambar 53 class Compressor.....	87
Gambar 54 class Disk	90
Gambar 55 class ExecutorServiceManager	91
Gambar 56 class File.....	93
Gambar 57 class module com.example.compactJV.....	96

