MultiThreadFileArchive Proje Raporu

1. Proje Tanımı

MultiThreadFileArchive, bir klasördeki .txt dosyalarını çoklu iş parçacığı (multithreading) ile analiz eden, arşivleyen (zip) ve arşivden çıkaran (unzip) bir Java uygulamasıdır. Program, her dosya için satır ve karakter sayımı yaparak istatistiksel bilgiler sunar. Tüm işlem sürelerini ölçerek performans raporu verir.

2. Temel Özellikler

- **Çoklu İş Parçacığı (Multithreading):** Her dosya farklı bir thread ile işlenerek paralel çalışma sağlanır.
- **Semaphore Kullanımı:** Aynı anda en fazla 10 thread çalışacak şekilde sınırlama yapılır.
- **Dosya Arşivleme ve Çıkarma:** İşlenen dosyalar zip formatında arşivlenir ve daha sonra unzip ile çıkarılır.
- **Zaman Takibi:** TimeLogger sınıfı ile tüm işlem süreleri ölçülür.
- **Güvenli Dosya Okuma:** SafeFileUtils ile dosya okuma hataları yakalanarak uygulama kararlılığı artırılır.

3. Dizin Yapısı

```
MultiThreadFileArchive/
├── input/              # .txt dosyalarının bulunduğu klasör
├── output/files.zip    # Arşivlenen çıktı dosyası
├── unzipped_output/    # Zip'ten çıkarılan dosyalar
├── src/
    ├── Main.java
    ├── model/FileStatistics.java
    ├── service/
    │   ├── FileAnalyzerService.java
    │   ├── FileReaderService.java
    │   └── ResultCollector.java
    ├── thread/
    │   ├── FileArchiverZip.java
    │   ├── FileArchiverUnzip.java
    │   ├── FileProcessorThread.java
    │   └── ThreadMonitor.java
    ├── util/SafeFileUtils.java
    └── worker/Worker.java
```

4. Kurulum ve Çalıştırma

Gereksinimler: Java 8 veya üzeri bir JDK

git clone https://github.com/smnrckr/MultiThreadFileArchive.git
cd MultiThreadFileArchive
javac -d out -sourcepath src src/Main.java
java -cp out Main

Not: input/ klasörüne .txt dosyaları eklemeyi unutmayın.

5. Görev Dağılımı

Semanur Çakır

- TimeLogger sınıfını projeye ekledim ve programda işlem sürelerini ölçmek için kullandım.
- SafeFileUtils yardımcı sınıfını yazdım; dosya okuma işlemlerini güvenli ve hataya dayanıklı hale getirdim.
- Dosya okuma işlemlerinde SafeFileUtils'i entegre ettim, böylece hata yönetimini güçlendirdim.
- Programın çeşitli noktalarında TimeLogger ile zaman ölçümü ve performans takibi sağladım
- Yorum satırlarıyla kodu diğer geliştiriciler için anlaşılır kıldım.

Elif Keleş

- FileArchiverZip ve FileArchiverUnzip sınıflarını Thread sınıfından kalıtım alarak yazdım.
- Ortak bir static değişken olarak BUFFER_SIZE tanımladım. Bu değeri dosyanın 4kb lik parçalar halinde I/O işlemlerinin yapılması için kullandım.
- Zip thread de öncelikle dosyanın akışından zip yazma akışına geçirerek zip entry ögeleri oluşturdum. Daha sonra bunlarla oluşturduğum zip i parametre olarak aldığım yola aktardım. Ve aktardıktan sonra inputtaki belgeleri silme işlemini yaptım.
- En sonda unzip thread ile zipten çıkararak unzipped klasöründe dosyaları kaydettim.

Mete Kerem Berk

- İnput klasörüne bakıp txt dosyası kadar thread açılmasını sağladım.
- Semaphore kullanarak bu thread sayısını 10 ile sınırladım.
- FileProcessorThread sınıfını Thread sınıfından kalıtım alarak yazdım ve bu thread sınıfının işlerini Worker sınıfında halledilmesini sağladım.
- Aynı anda kaç thread kullanıldığını görebilmek için ThreadMonitor sınıfıyla thread sayısını tuttum.

Fatma Nur Kurt

- FileStatistics, FileAnalyzerService, FileReaderService ve Worker sınıflarını yazarak dosya analiz sürecinin temelini oluşturdum. Bu sınıflar ile .txt dosyalarındaki satır ve karakter sayılarının hesaplanmasını sağladım.
- Worker sınıfında tek bir dosyanın analizini yönettim.
- Projenin konsol çıktılarını daha anlaşılır hale getirdim. Tüm System.out.println() mesajlarını okunabilirlik ve kullanıcı deneyimi açısından yeniden düzenledim.
- Kodun her satırına işlevini anlatan yorum satırları ekleyerek ekip üyelerinin projeyi daha rahat anlamasını sağladım.
- Ayrıca, development ve feature branch'leri birleştirirken oluşan merge conflict'leri çözdüm.

Ömer Gün

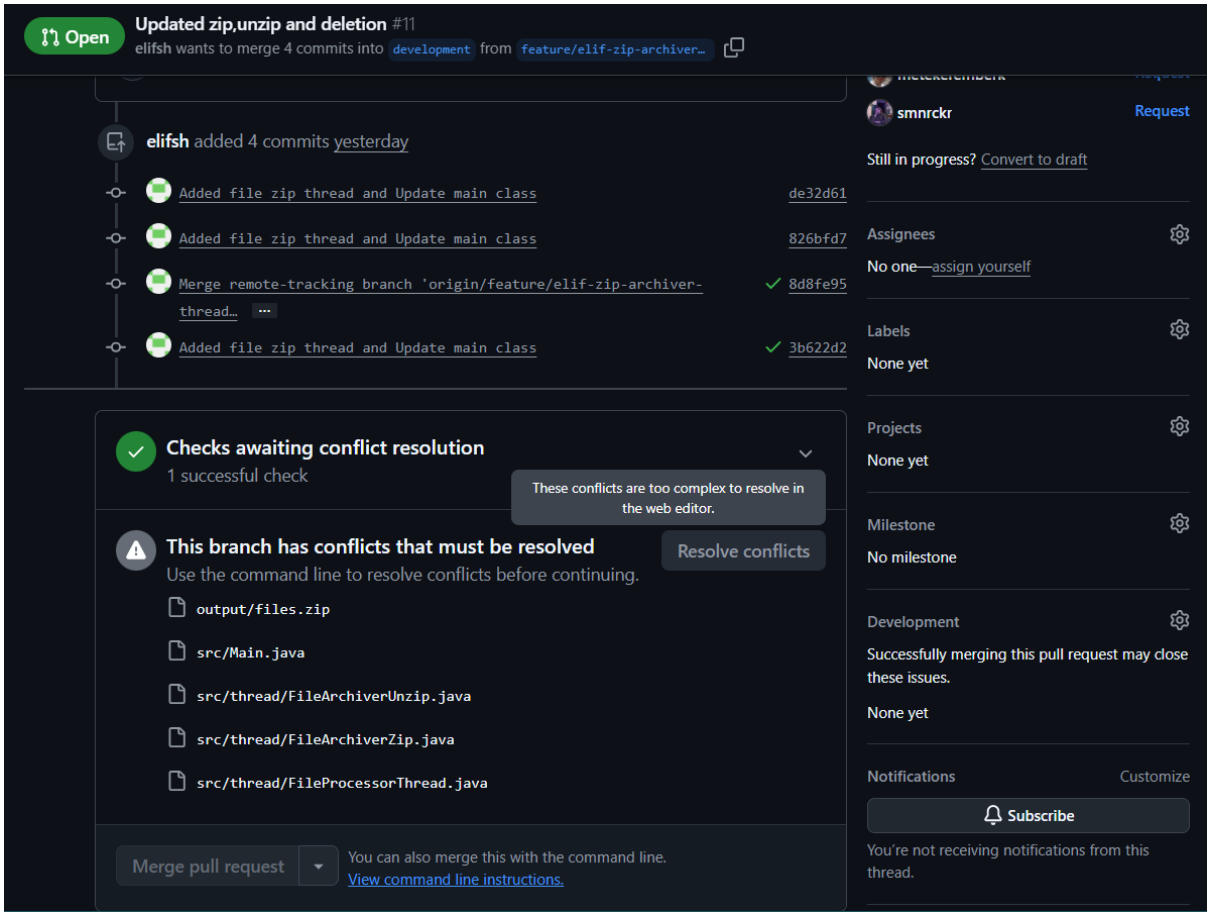
- FileArchiverZip sınıfına satır ve karakter sayısını hesaplayan analiz işlemi entegre edildi.
- Mevcut FileAnalyzerService ve FileStatistics sınıfları kullanılarak her dosya analiz edildi.
- Analiz sonuçları, sınıf içinde tanımlı ConcurrentHashMap<String, FileStatistics> üzerinden thread-safe şekilde saklandı.
- Böylece ResultCollector gibi harici bir yapıya gerek kalmadan sonuçlar doğrudan bu sınıfta toplandı.
- Çoklu iş parçacığı (multithreading) ortamında veri bütünlüğü ve güvenliği sağlandı.
- Kodun sorumluluk dağılımı sadeleştirildi, analiz ve zipleme işlemi aynı akış içinde birleştirildi.
- Kodun okunabilirliği ve modülerliği artırılarak bakım kolaylaştırıldı.

6. Pull Request ve Conflict

Geliştirme tamamlandıktan sonra feature/elif-zip-archiver-thread branch'inden development branch'ine pull request (PR) oluşturuldu. Pull request sırasında GitHub, aşağıdaki dosyalarda otomatik birleştirme yapılamayacağını belirtti ve çatışma çözümü istendi:

- output/files.zip
- src/Main.java
- src/thread/FileArchiverUnzip.java
- src/thread/FileArchiverZip.java
- src/thread/FileProcessorThread.java

GitHub arayüzünde "This branch has conflicts that must be resolved" uyarısı gösterildi. Bu durum, .zip gibi binary dosyaların ve aynı dosya üzerinde yapılan paralel değişikliklerin çakışmasından kaynaklandı.

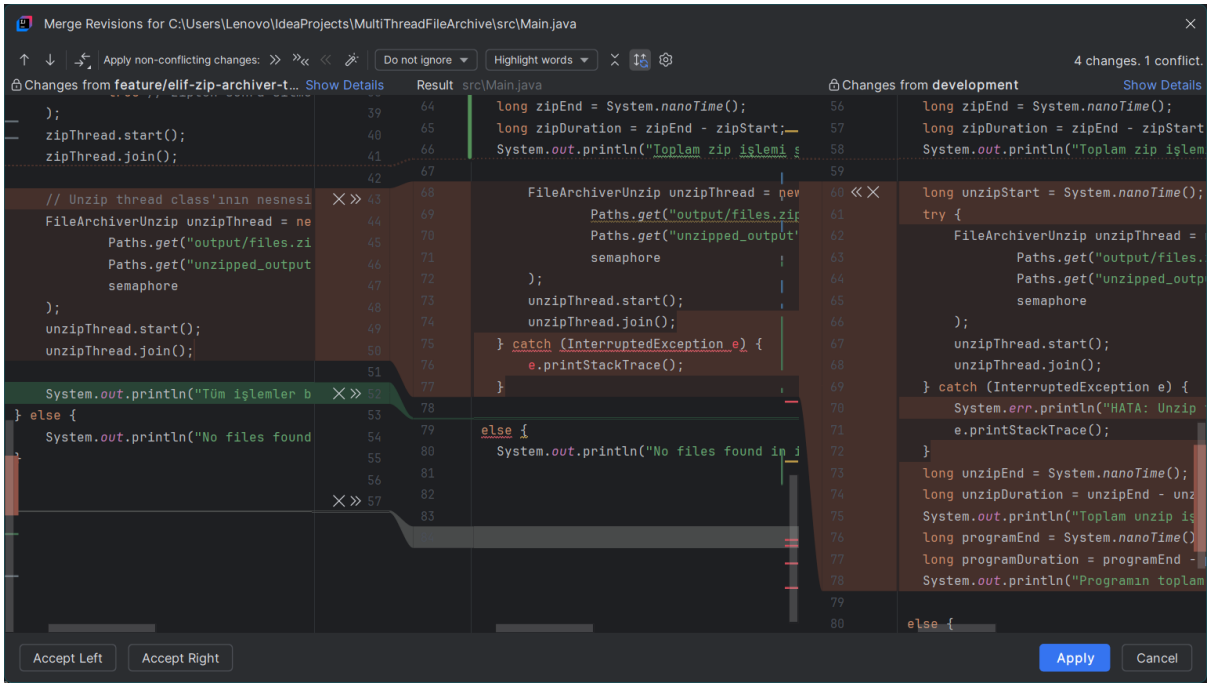


Şekil 1: GitHub PR sayfasında conflict uyarısı

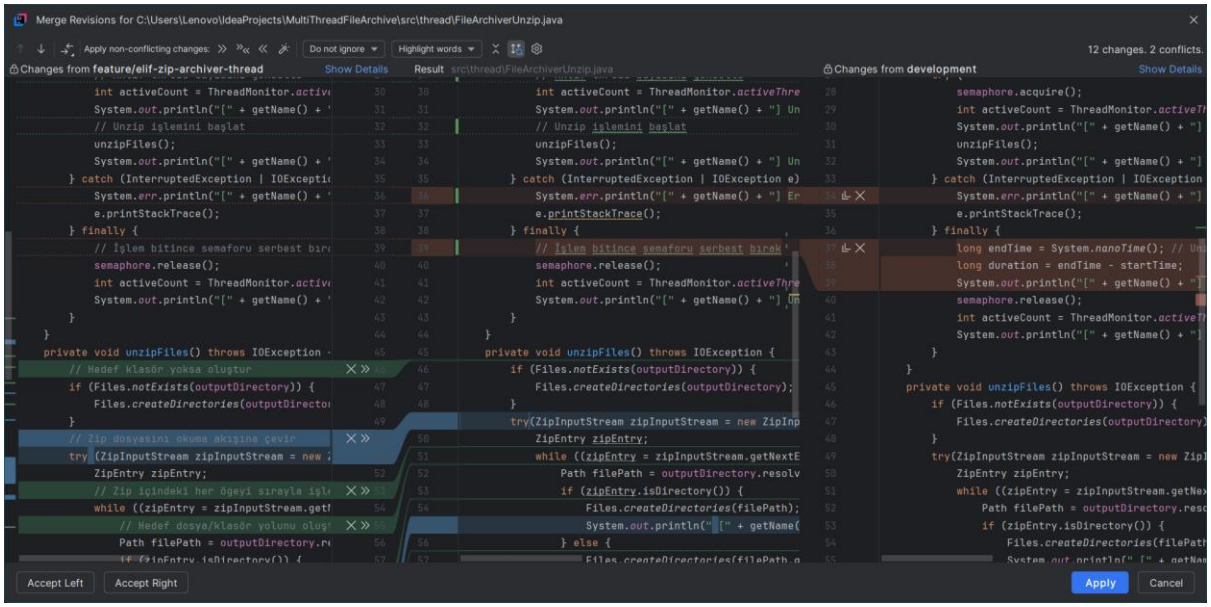
Conflict Çözüm Süreci

Çatışmalar lokal ortamda (IntelliJ IDEA) çözüldü ve ilgili değişiklikler commit'lenerek PR başarılı şekilde güncellendi.

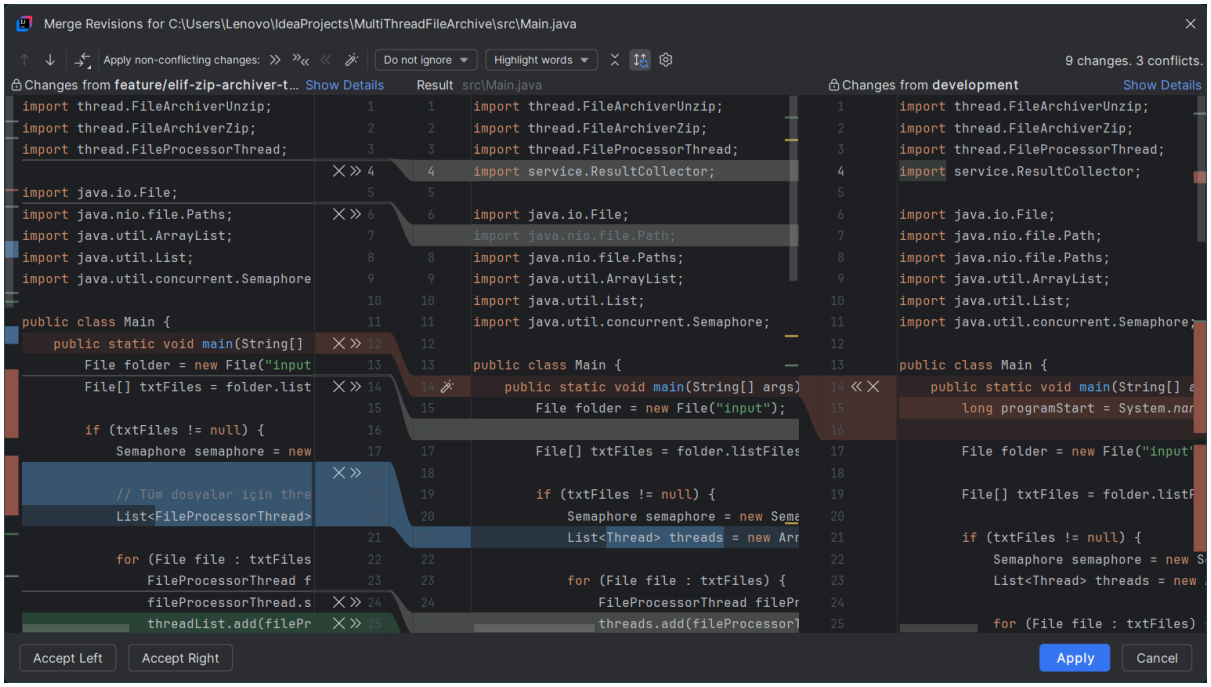
Resolved by Fatma Nur Kurt, Reviewed by Elif Keleş



Şekil 2: Örnek Conflict Çözüm Ekranı-1

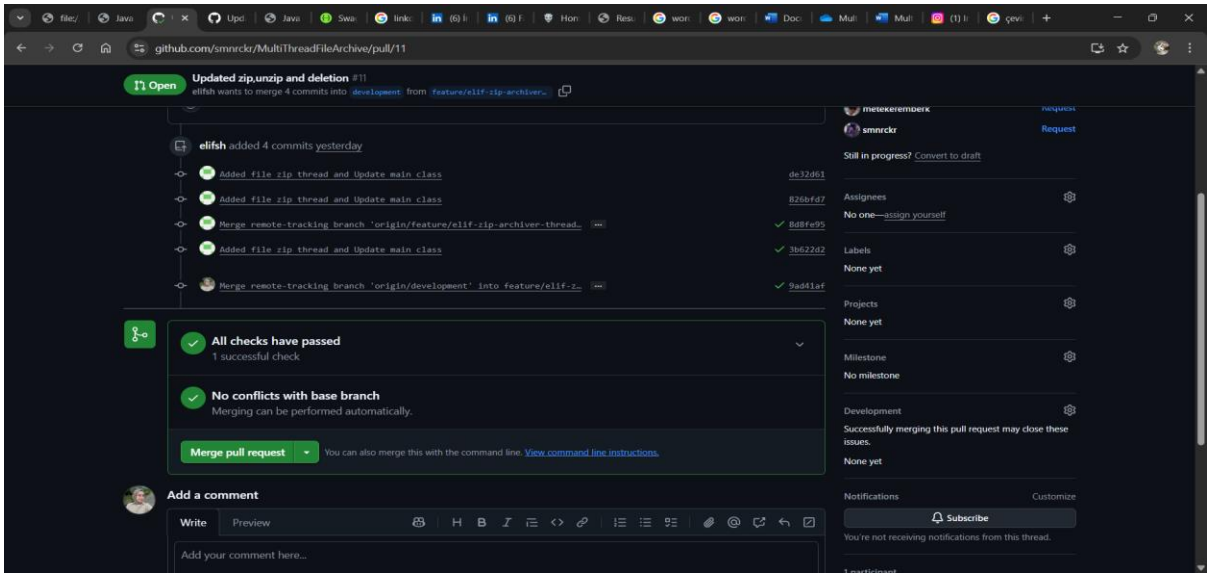


Şekil 3: Örnek Conflict Çözüm Ekranı-2



Şekil 4: Örnek Conflict Çözüm Ekranı-3

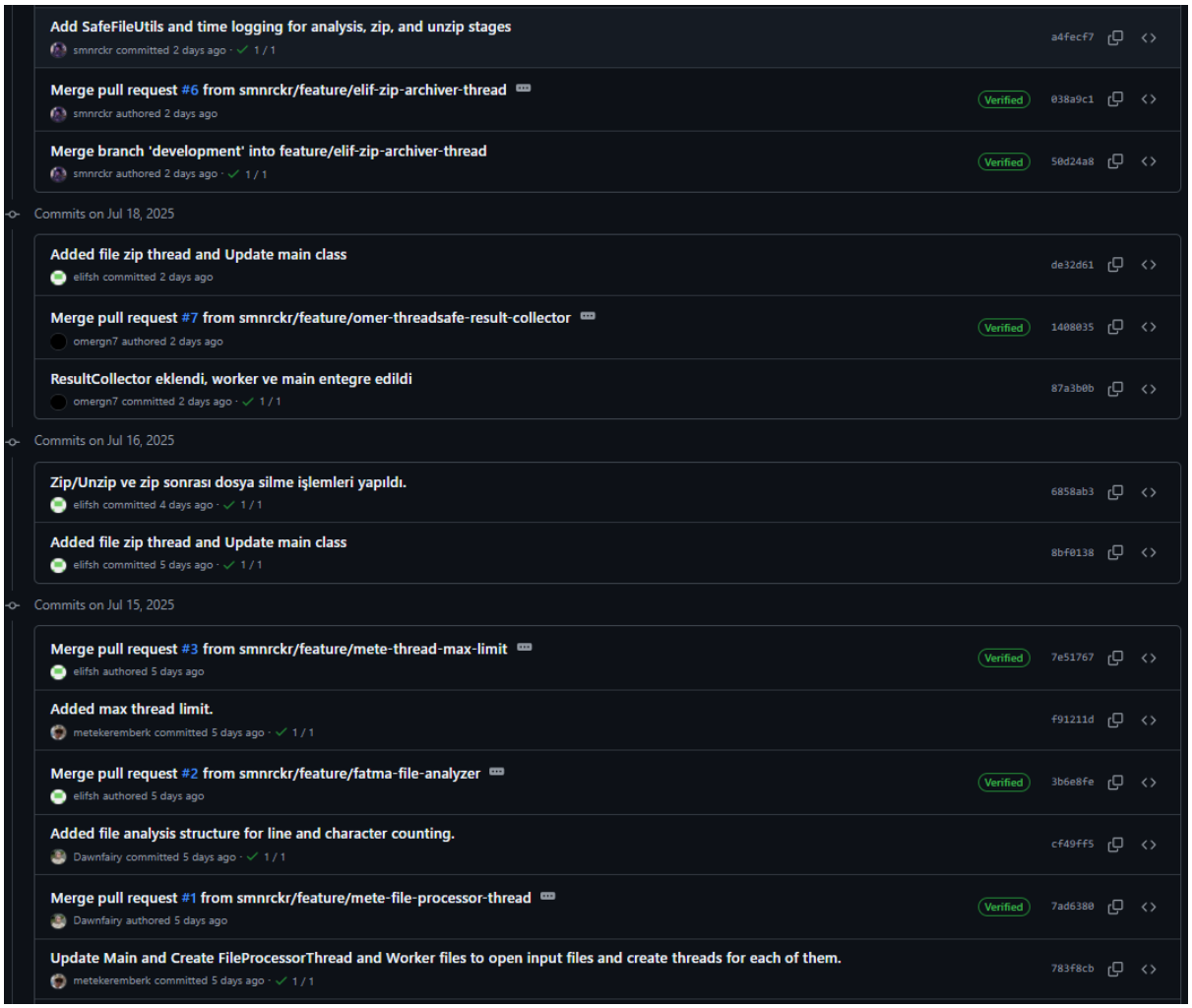
Sonrasında GitHub üzerinden PR merge edilebilir duruma geldi.



Şekil 5: PR çözüm sonrası merge edilebilir duruma gelmiş ekran

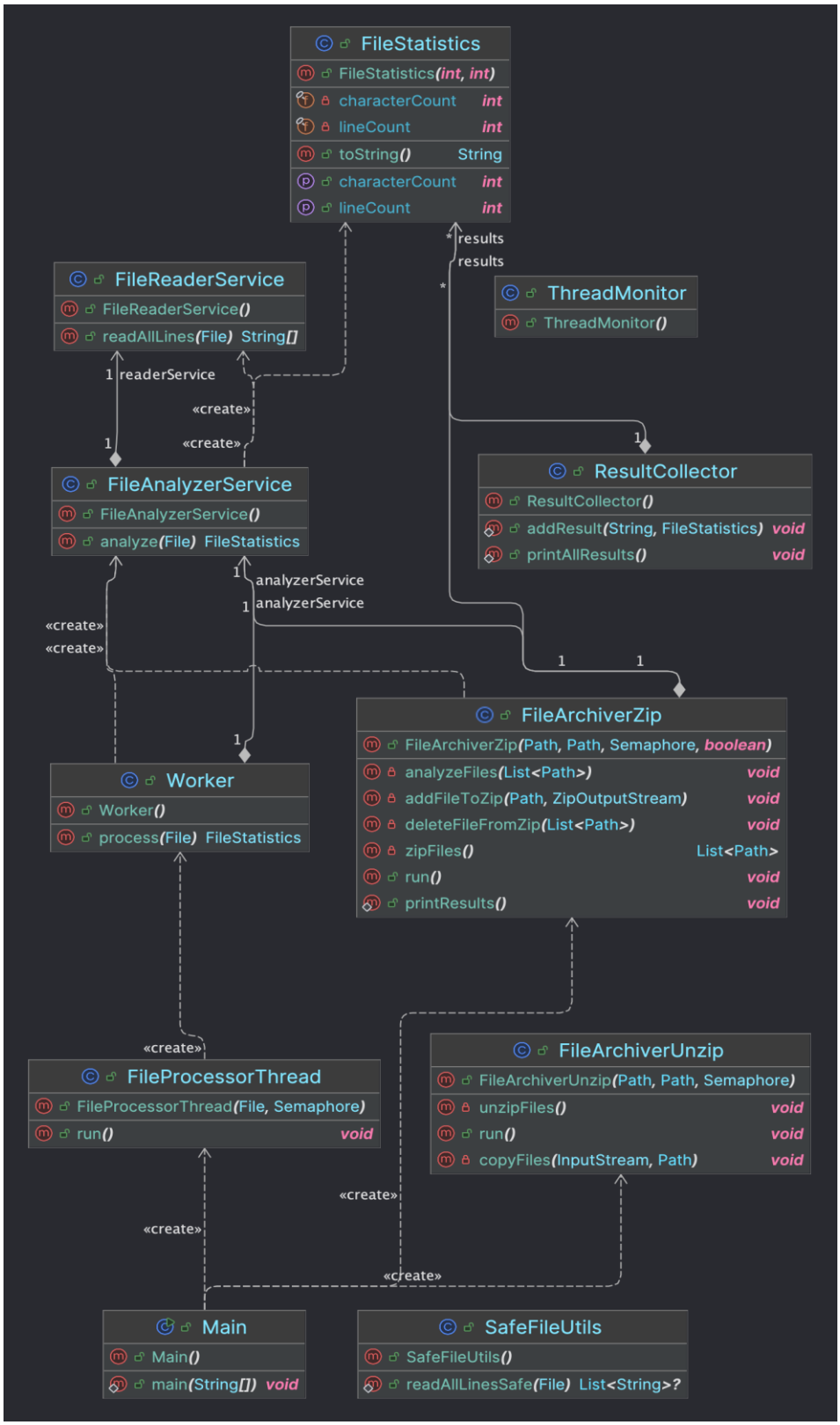
7. Sınıf ve Paket Açıklamaları

- **Main.java:** Uygulamanın ana akışını başlatır.
- **model.FileStatistics:** Satır ve karakter sayısını tutar.
- **service.FileAnalyzerService:** Dosya analizini yapar.
- **service.FileReaderService:** Dosyayı okur ve satırlara ayırır.
- **service.ResultCollector:** Sonuçları toplar.
- **thread.FileProcessorThread:** Her dosya için thread oluşturur.
- **thread.FileArchiverZip/Unzip:** Zip ve unzip işlemlerini yürütür.
- **thread.ThreadMonitor:** Aktif thread sayısını takip eder.
- **util.SafeFileUtils:** Hataları yakalayan dosya okuma yöntemlerini sunar.
- **worker.Worker:** Dosya analizini yapar ve sonucu toplar.

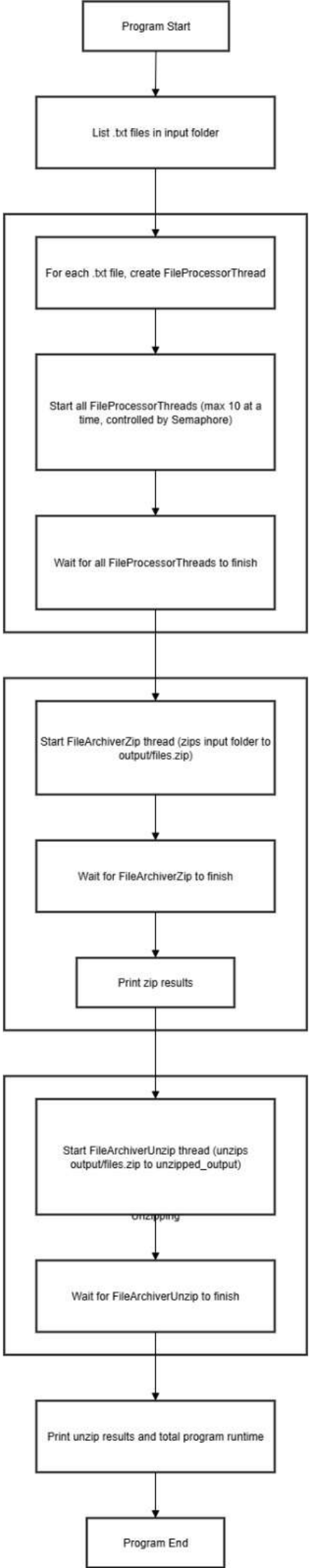


Şekil 6: Örnek Commit Geçmişi

8. Class Diyagramı ve Akış Şeması



Şekil 7: Class Diyagramı


Program Start
List .txt files in input folder
For each .txt file, create FileProcessorThread
Start all FileProcessorThreads (max 10 at a time, controlled by Semaphore)
Wait for all FileProcessorThreads to finish
Start FileArchiverZip thread (zips input folder to output/files.zip)
Wait for FileArchiverZip to finish
Print zip results
Start FileArchiverUnzip thread (unzips output/files.zip to unzipped_output)
Wait for FileArchiverUnzip to finish
Print unzip results and total program runtime
Program End