Java Multithread Dosya Analiz ve Arşivleme Projesi

Grup Proje Raporu

Teslim Tarihi: 20 Temmuz 2025

Takım Üyeleri:
Şevval Kaşan, Hakan Atalay, Umut Bayar , Bayram Kaya , Furkan Can

Contents

1 Proje Amacı

Bu projenin amacı, Java dilinde çoklu iş parçacığı (multithreading) konseptini uygulamalı bir şekilde deneyimlemek ve eşzamanlı çalışan thread'ler arasında veri paylaşımını güvenli hale getirmektir. Proje kapsamında birden fazla metin dosyasının aynı anda analiz edilmesi, analiz sonuçlarının thread-safe bir veri yapısında saklanması, tamamlanan verilerin ayrı bir thread tarafından arşivlenmesi ve tüm sürecin zaman ölçümlerinin yapılması hedeflenmiştir.

2 Proje Konusu ve Akış

Projede birden fazla metin dosyası paralel olarak analiz edilmekte, sonrasında ayrı bir thread bu dosyaları zip arşivine toplamaktadır. Tüm işlemlerin süresi ölçülmekte ve sonuçlar ekrana yazdırılmaktadır.

2.1 Akış Adımları

1. Program başlatılır.
2. input/ klasörü taranır ve .txt dosyaları listelenir.
3. Her dosya için ayrı bir Thread başlatılır.
4. Thread'ler dosya analizini yapar ve sonuçları ConcurrentHashMap'e ekler.
5. Ana thread, join() ile tüm thread'lerin tamamlanmasını bekler.
6. ArchiveTask thread'i başlatılır.
7. Tüm dosyalar output/archived-files.zip olarak arşivlenir.
8. Süre ölçümleri ekrana yazdırılır.
9. (Opsiyonel) .txt dosyaları silinir.
10. Program sonlanır.

2.2 Proje Akış Diyagramı

Proje süreci, aşağıdaki akış diyagramında görsel olarak ifade edilmiştir. Bu diyagramda, programın başlatılmasından dosyaların analizine, sonuçların saklanmasından arşivleme işlemi ve programın sonlandırılmasına kadar tüm adımlar gösterilmektedir.

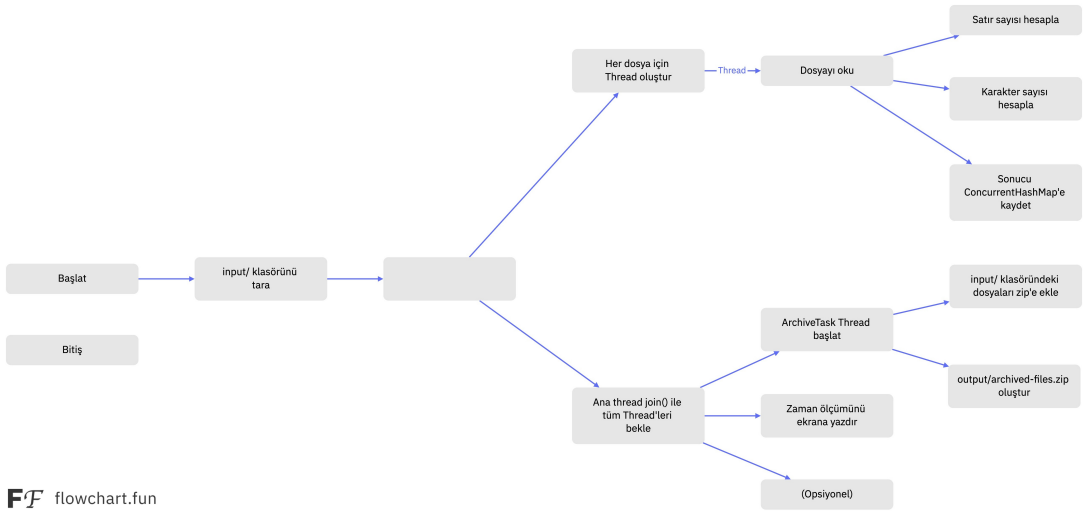

Figure 2.1: Proje Akış Diyagramı

2.3 Sınıflar ve Görevleri

Projede görev paylaşımı yapılmış ve her sınıf belirli bir sorumluluğu yerine getirmek üzere tasarlanmıştır. Aşağıdaki tabloda sınıfların işlevleri özetlenmiştir:

Sınıf	**Açıklama**
`Main`	Uygulamanın başlangıç noktasıdır. `input/` klasöründeki dosyaları tespit eder, her dosya için bir iş parçacığı başlatır, tüm analizlerin tamamlanmasını bekler ve ardından arşivleme sürecini tetikler.
`FileAnalysisTask`	`Runnable` arayüzünü implemente eden bu sınıf, her dosya için ayrı bir iş parçacığında çalışır. Dosyanın satır ve karakter sayısını hesaplayarak sonucu merkezi veri yapısına ekler.
`ArchiveTask`	Dosya analizlerinin ardından çalışan ayrı bir iş parçacığıdır. Tüm `.txt` dosyalarını `output/archived-files.zip` dosyasına sıkıştırarak arşivler.
`FileStats`	Her bir dosyanın istatistiklerini (satır ve karakter sayısı) tutan veri modelidir. Bu model, `ConcurrentHashMap` içinde değer olarak kullanılmaktadır.

Table 2.1: Projede Kullanılan Sınıfların Görevleri

2.4 Kullanılan Veri Yapıları

Projede çoklu iş parçacığı ile çalışan sınıflar arasında güvenli veri paylaşımı sağlamak amacıyla `ConcurrentHashMap<String, FileStats>` veri yapısı tercih edilmiştir.

`ConcurrentHashMap`, Java'nın `java.util.concurrent` paketinde yer alan ve eşzamanlı (concurrent) işlemler için optimize edilmiş bir harita (map) yapısıdır. Bu yapı sayesinde birden fazla iş parçacığı aynı anda veri ekleyebilir, okuyabilir veya güncelleyebilir. Geleneksel `HashMap` sınıfının aksine, eşzamanlı erişimlerde veri tutarsızlığı veya `ConcurrentModificationE` gibi hatalar oluşmaz.

Projedeki kullanım amacı:

- Her `FileAnalysisTask` nesnesi, işlediği dosyanın istatistiklerini (satır ve karakter sayısı) bu yapıya kaydeder.
- Aynı anda çalışan birden fazla thread, `ConcurrentHashMap` üzerinde güvenli şekilde işlem yapabilir.
- Sonuçlar, ana thread tarafından merkezi olarak erişilerek raporlamada kullanılır.

Bu veri yapısı sayesinde senkronizasyon problemi yaşanmadan yüksek performanslı ve güvenli bir analiz süreci gerçekleştirilmiştir.

2.5 Hata Yönetimi

Projede karşılaşılabilecek hatalar öngörülerek, sistemin stabilitesini korumak amacıyla kapsamlı bir hata yönetimi stratejisi uygulanmıştır. Hatalar `try-catch` blokları ile yakalanmış, kullanıcı bilgilendirilmiş ve kritik hatalarda programın güvenli bir şekilde sonlandırılması sağlanmıştır.

- **Dosya Okuma Hataları:** Her `FileAnalysisTask` içinde dosya okuma işlemleri sırasında `IOException` yakalanmaktadır. Dosya erişilemezse, ilgili dosya atlanmakta ve sistem diğer dosyalarla devam etmektedir. Bu sayede tek bir dosyadaki sorun tüm süreci etkilememektedir.
- **Arşivleme Hataları:** `ArchiveTask` sınıfı içerisinde `ZipOutputStream` işlemleri sırasında oluşabilecek hatalar yakalanarak kullanıcıya bilgi verilir. Arşivleme başarısız olursa program kontrollü biçimde sonlandırılır.
- **Genel Hata Mesajları:** Hatalar anlaşılır ve açıklayıcı mesajlarla `System.err.println()` kullanılarak loglanmakta, hata ayıklama süreci kolaylaştırılmaktadır.
- **Thread Hataları:** `InterruptedException` gibi eşzamanlılık sırasında oluşabilecek özel durumlar da ilgili noktalarda ele alınmıştır. Böylece ana thread ile alt thread'ler arasında sağlıklı bir kontrol sağlanmıştır.

Bu yaklaşım sayesinde sistem, beklenmeyen durumlara karşı dayanıklı hale getirilmiş ve kullanıcı deneyimi bozulmadan süreç yönetimi sağlanmıştır.

2.6 Örnek Çıktılar

Aşağıda, programın örnek bir çalıştırması sırasında elde edilen konsol çıktısı yer almaktadır. Dosya analiz süreci, arşivleme işlemi, zaman ölçümleri ve başarılı tamamlanma mesajları detaylı olarak görülmektedir.

```
Program başlatılıyor
Dosya analizi başlatılıyor
Example3.txt - 9 satır / 396 karakter
Example1.txt - 3 satır / 195 karakter
Example2.txt - 8 satır / 320 karakter

Toplam: 20 satır / 911 karakter

Arşivleme başlatılıyor
Dosyalar başarıyla ziplenmiştir: output\archived-files.zip
Arşivleme süresi: 40 ms

Raporlama yapılıyor
Example3.txt - 9 satır / 396 karakter
Example1.txt - 3 satır / 195 karakter
Example2.txt - 8 satır / 320 karakter

Toplam: 20 satır / 911 karakter
Program tamamlandı. Toplam süre: 0,092 saniye
```

2.7 Örnek Kod Parçası

Aşağıda, dosya analiz görevini üstlenen `FileAnalysisTask` sınıfının `run()` metoduna ait örnek bir kod parçası sunulmuştur. Bu metot, dosyayı satır satır okuyarak satır sayısını ve toplam karakter miktarını hesaplamaktadır. Elde edilen istatistikler, thread-safe yapı olan `ConcurrentHashMap` içerisine kaydedilir.

```
1  @Override
2  public void run() {
3      try (BufferedReader reader = new BufferedReader(new FileReader(file)
       )) {
4          String line;
5          int lineCount = 0;
6          int charCount = 0;
7
8          while ((line = reader.readLine()) != null) {
9              lineCount++;
10             charCount += line.length();
11         }
12
```

```
13          FileStats stats = new FileStats(lineCount, charCount);
14          resultMap.put(file.getName(), stats);
15      } catch (IOException e) {
16          System.err.println("Dosya okunamad : " + file.getName());
17      }
18  }
```

Listing 2.1: FileAnalysisTask.java – run() Metodu

2.7.1 Zaman Ölçümü

Projede, her bir işlemin süresi detaylı olarak ölçülmüş ve sonuçlar kullanıcıya sunulmuştur. Bu ölçümler, programın performansını analiz etmek ve darboğazları tespit etmek açısından önemlidir.

Zaman ölçümlerinde Java'nın yüksek çözünürlüklü zaman fonksiyonu olan `System.nanoTime()` kullanılmıştır.

- Dosya analiz süresi
- Arşivleme süresi
- Programın toplam çalıştırma süresi

```
1  long startTime = System.nanoTime();
2
3  // i lem yap l r
4
5  long endTime = System.nanoTime();
6  double totalTimeSeconds = (endTime - startTime) / 1e9;
7  System.out.printf("Toplam s re: %.3f saniye%n", totalTimeSeconds);
```

Listing 2.2: Zaman Ölçüm Örneği

3 Ekran Görüntüleri

Bu bölümde, projenin çalıştırılması ve sürüm kontrol süreçlerine ait ekran görüntüleri sunulmuştur. Görseller, uygulamanın nasıl çalıştığını ve Git üzerinden nasıl organize edildiğini görsel olarak desteklemektedir.

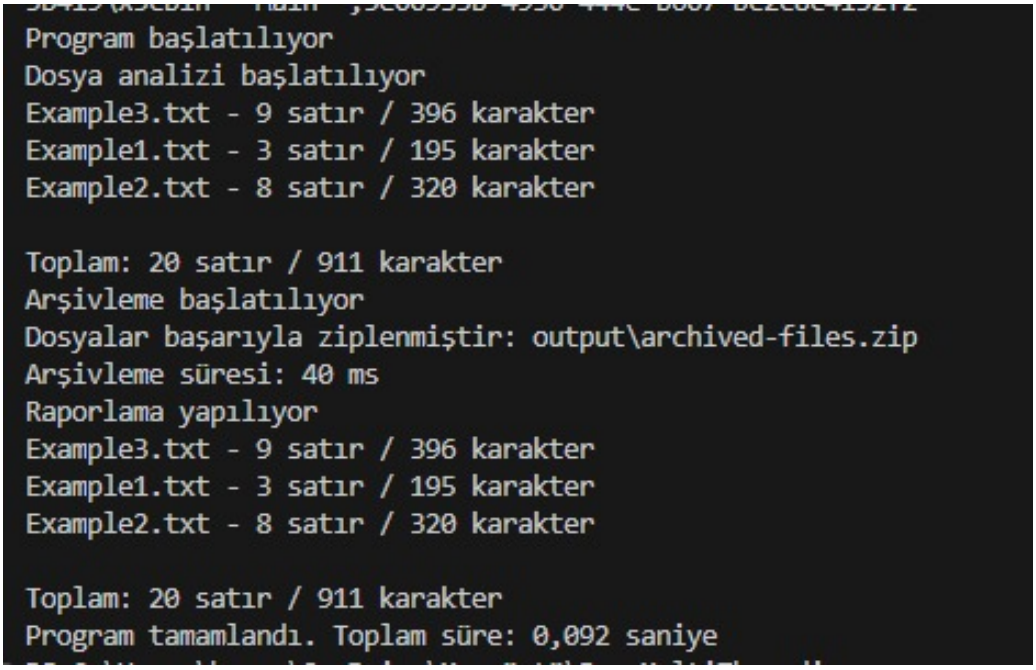

Figure 3.1: Uygulamanın konsol çıktısı: Dosya analizi, arşivleme ve süre bilgileri

4 İş Bölümü

Projede görev paylaşımı yapılarak her ekip üyesi belirli modüllerden sorumlu olmuştur. Aşağıda ekip üyeleri ve sorumluluk alanları detaylı biçimde listelenmiştir:

- **Şevval Kasan – Raporlama ve Belgelendirme**
  - Proje raporunun hazırlanması ve düzenlenmesi
  - Konsol çıktılarının raporlanması ve açıklamaların yazımı
- **Bayram – Thread Yönetimi**
  - `FileAnalyzer.java` ve `FileAnalysisTask.java` dosyalarının geliştirilmesi
  - `input/` klasörünü tarayan yapının oluşturulması
  - Her dosya için ayrı thread oluşturulması ve `join()` işlemlerinin tasarlanması
- **Furkan – Arşivleme Sorumlusu**
  - `ArchiveTask.java` dosyasının yazılması
  - Analizlerin tamamlanmasını bekleyen arşivleme thread'inin geliştirilmesi

  - Dosyaların `.zip` formatında arşivlenmesi
  - Hata kontrol mekanizmalarının ve kullanıcı bilgilendirme mesajlarının eklenmesi

- **Umut – Program Akışı ve Zaman Ölçümü**
  - `Main.java` ve `Utils.java` dosyalarının oluşturulması
  - Programın giriş noktası (`main` metodu) ve genel akışın kontrolü
  - `System.nanoTime()` ile aşama sürelerinin ölçülmesi

- **Hakan – Veri Modelleme ve Sonuç Yapısı**
  - `FileStats.java` sınıfının yazılması (satır ve karakter sayısı verilerini tutar)
  - `ConcurrentHashMap` ile thread-safe veri yapısının oluşturulması
  - Analiz sonuçlarının kullanıcıya düzenli şekilde sunulması

5 README.md İçeriği

Projeye ait GitHub deposunda yer alan `README.md` dosyası, projenin anlaşılabilirliğini artırmak ve dışarıdan erişen geliştiricilerin projeyi kolayca çalıştırabilmesini sağlamak amacıyla hazırlanmıştır. Dosyada şu bilgiler yer almaktadır:

- **Projenin Amacı ve Tanımı:** Java dilinde çoklu iş parçacığı (multithreading) kullanılarak dosya analizi ve arşivleme işlemlerinin nasıl gerçekleştirildiği anlatılmıştır.
- **Kurulum ve Çalıştırma Adımları:** Gerekli klasörlerin oluşturulması, örnek `.txt` dosyalarının `input/` klasörüne eklenmesi ve `Main.java` sınıfının nasıl çalıştırılacağı adım adım açıklanmıştır.
- **Örnek Çıktılar:** Konsol ekranında görülen analiz sonuçları, toplam satır/karakter bilgileri ve arşivleme çıktıları örneklerle sunulmuştur.
- **İş Bölümü:** Takım üyelerinin hangi modüllerden sorumlu olduğu listelenmiş, ekip çalışmasının nasıl organize edildiği gösterilmiştir.
- **Akış Diyagramı:** Proje işleyişini açıklayan akış diyagramı görsel olarak eklenmiştir. Bu diyagram, proje akışını hızlıca kavramak isteyen kullanıcılar için özet bir rehber sunmaktadır.

5.1 Nasıl Çalıştırılır

Projeyi kendi bilgisayarınızda çalıştırmak için aşağıdaki adımları izleyebilirsiniz. Proje Java diliyle geliştirilmiştir, bu nedenle Java Development Kit (JDK 11 veya üzeri) sisteminizde kurulu olmalıdır.

1. **Projeyi GitHub üzerinden klonlayın:**

```
git clone https://github.com/kullaniciadi/projeadi.git
cd projeadi

```

2. **`input/` klasörünü oluşturun ve içine `.txt` dosyaları ekleyin:**

   - Her bir `.txt` dosyası ayrı bir thread tarafından analiz edilecektir.
   - Örnek dosyalar: `example1.txt`, `example2.txt`, vb.

3. **Proje dosyalarını derleyin ve çalıştırın:**

```
javac Main.java
java Main

```

   Alternatif olarak bir Java IDE (örneğin IntelliJ IDEA veya Eclipse) üzerinden `Main.java` dosyasını çalıştırabilirsiniz.

4. **Program çalıştıktan sonra:**

   - Konsol çıktısında her dosyanın satır ve karakter sayısı gösterilir.
   - `output/archived-files.zip` dosyası oluşturulur.

6 Sonuç

Bu proje multithread kullanımı, eşzamanlı veri yönetimi ve arşivleme süreçlerinin uygulanmasını sağlamış; kod kalitesi, Git versiyon kontrolü ve ekip çalışması gibi önemli kazanımlar sunmuştur.