DOSYA ARŞİVLEYİCİ

HÜSEYİN ÇAĞLAR

Proje Tanıtımı

Bu proje, büyük dosyaların sıkıştırılması ve çıkarılması işlemlerini hızlı ve verimli bir şekilde gerçekleştirmeyi hedeflemektedir. Başlangıçta, her dosya için yeni bir iş parçacığı (thread) başlatan asenkron bir yaklaşım kullanılıyordu. Ancak, işlem sürelerini daha da kısaltmak ve kaynak kullanımını optimize etmek amacıyla thread pool yöntemi entegre edilmiştir. Thread pool, bir dizi önceden oluşturulmuş iş parçacığını kullanarak, gereksiz iş parçacığı oluşturma işleminden kaçınır ve bu sayede daha verimli bir kaynak yönetimi sağlar. Bu, özellikle büyük dosya arşivleme işlemleri için önemli bir avantaj sağlamaktadır.

Arka Plan

Projede, dosya sıkıştırma ve çıkarma işlemleri asenkron bir şekilde başlatılıyordu. Bu işlem, her dosya için ayrı bir iş parçacığı başlatmayı içeriyordu. Ancak, bu yaklaşım büyük dosyaların yönetiminde verimsizliklere yol açıyordu. Dosya arşivleme işlemleri sırasında çok sayıda iş parçacığının paralel çalışması kaynak kullanımında sorunlar yaratabiliyordu. Bu yüzden, thread pool yöntemi benimsenmiştir. Thread pool, çok sayıda iş parçacığı oluşturmak yerine, önceden belirlenmiş sayıda iş parçacığı kullanarak işlemleri daha verimli hale getirmektedir. Ayrıca, thread pool sayesinde her dosya için yeni bir iş parçacığı başlatma ihtiyacı ortadan kalkmakta ve bu da performansı önemli ölçüde iyileştirmektedir.

Problem Tanımı

Asenkron işlem yaklaşımı, her dosya için yeni bir iş parçacığı başlatmakta ve bu işlem zamanla çok sayıda iş parçacığına yol açmaktadır. Bu durum, büyük dosya arşivleme işlemleri için verimsiz hale gelmektedir. Her dosya için yeni bir iş parçacığı başlatmak, işlem sürelerinin uzamasına ve kaynak kullanımının artmasına neden olmaktadır. Bu verimsizlikler, proje için ciddi bir problem oluşturuyordu. Asenkron yaklaşımda, iş parçacıklarının fazla sayıda olması, işlem sürelerini artırmakta ve sistemin performansını düşürmektedir. Bu sebeple thread pool yaklaşımına geçiş yapılmıştır. Thread pool yöntemi, önceden oluşturulmuş iş parçacıkları havuzunu kullanarak işlemleri daha verimli hale getirmekte ve kaynak kullanımını optimize etmektedir.

Projenin Amacı

Bu projenin temel amacı, dosya sıkıştırma ve çıkarma işlemlerini daha verimli bir hale getirmek için thread pool teknolojisinin kullanılmasıdır. Asenkron işlemler yerine thread pool kullanarak, çok sayıda iş parçacığının daha verimli bir şekilde yönetilmesi hedeflenmektedir. Bu yöntemle, özellikle büyük dosyaların işlenmesinde işlem sürelerinin önemli ölçüde kısalması ve kaynak kullanımının azalması beklenmektedir. Thread pool, sistemdeki kaynakları daha verimli kullanarak, daha hızlı ve etkili bir işlem süreci sağlamaktadır.

Kapsam

Bu proje yalnızca dosya sıkıştırma ve çıkarma işlemlerini kapsamaktadır. Ancak, thread pool teknolojisi sayesinde büyük dosyaların işlenmesinde önemli zaman tasarrufu sağlanacaktır. Proje, büyük veri setlerinin hızlı bir şekilde işlenmesini sağlayarak, özellikle büyük ölçekli projeler için etkili bir çözüm sunmaktadır. Çoklu iş parçacıkları yöneten bir sistem gereksinimi olan tüm kullanıcılar için kullanılabilir bir yapı sunulmaktadır.

Hedefler

Performans İyileştirme: Thread pool kullanarak, dosya sıkıştırma ve çıkarma işlemlerinde çoklu iş parçacığı yönetimini daha verimli hale getirmek ve işlem sürelerini azaltmak.

Büyük Dosya Desteği: Büyük boyutlu dosyaların işlenmesinde daha etkili bir çözüm sunmak.

Kullanıcı Deneyimi: Basit ve kullanıcı dostu bir arayüz ile işlemleri kolaylaştırmak.

Beklenen Çıktılar

Thread pool yöntemiyle hızlandırılmış dosya sıkıştırma ve çıkarma işlemleri.

Daha verimli iş parçacığı yönetimi sayesinde kullanıcıların daha hızlı ve düşük kaynak tüketen bir deneyim elde etmeleri.

Dosya yönetimi süreçlerinde genel verimlilik artışı.

Gereksinim Analizi

Kullanıcı Senaryoları

Senaryo 1: *Dosya Sıkıştırma İşlemi Başlatma*
Kullanıcı, bir dosya sıkıştırma işlemi başlattığında, uygulama thread pool teknolojisini kullanarak işlemi hızla tamamlar. Bu, birden fazla iş parçacığının paralel çalışmasını sağlayarak işlem süresini önemli ölçüde kısaltır. Kullanıcı, işlemin daha hızlı bir şekilde tamamlandığını görür ve bu sayede zaman kazancı sağlar.

Senaryo 2: *Sıkıştırılmış Dosya Çıkarma*
Kullanıcı, sıkıştırılmış bir dosyayı çıkarmak istediğinde, işlem thread pool kullanılarak paralel işleme yapılır. Bu özellikle büyük dosyaların yönetilmesinde işlem sürelerini optimize eder. Thread pool ile yapılan bu işlem, asenkron olarak çalıştığı için daha verimli ve hızlı sonuçlar elde edilir.

Senaryo 3: *İşlem Durumu ve Sonucu Geri Bildirimi*
Kullanıcı, işlemin durumunu ve sonucunu proje üzerinden izleyebilir. Thread pool yöntemi ile işlem süresindeki kısalma sayesinde, kullanıcı daha hızlı bir geri bildirim alır. Bu da kullanıcı deneyimini iyileştirir ve işlemin başarılı olup olmadığına dair anında bilgi sağlar.

Gereksinimlerin Önceliklendirilmesi

Yüksek Öncelik:

Thread pool desteğiyle işlem sürelerinin kısaltılması: Multithreading teknolojisinin thread pool yöntemi ile optimize edilmesi, kaynakları daha verimli kullanarak işlem sürelerini önemli ölçüde azaltır. Bu özellik, projenin en kritik gereksinimlerinden biridir ve kullanıcıya hızlı sonuçlar sağlar.

Orta Öncelik:

Kullanıcı arayüzünün basit ve kullanıcı dostu olması: Thread pool kullanımı arka planda verimli bir şekilde çalışırken, kullanıcı arayüzü anlaşılır ve basit olmalıdır. Kullanıcı deneyimini olumsuz etkilemeden hızlı sonuçlar sunulmalıdır.

Düşük Öncelik:

İşlem sırasında ilerleme çubuğu ve işlem detaylarının sunulması: Thread pool yönteminin sağladığı hız, ilerleme çubuğunun daha hızlı güncellenmesini ve kullanıcıya daha dinamik bir deneyim sunulmasını sağlar. Bu, kullanıcıyı bilgilendiren bir ek özellik olarak yer alabilir.

Kabul Kriterleri

Dosya Sıkıştırma ve Çıkarma İşlemleri Başarıyla Tamamlanmalıdır:
Thread pool kullanılarak yapılan işlem süreleri, geleneksel tek thread yöntemine göre gözle görülür şekilde daha hızlı olmalıdır. Performans testlerinde, thread pool kullanımı ile işlem süresinde belirgin bir azalma gözlemlenmelidir.

Multithreading ile İşlem Sürelerinin İyileştirilmesi:
Thread pool kullanımı sayesinde, her işlem için yeni iş parçacıkları oluşturulması yerine, önceden oluşturulmuş iş parçacıkları havuzundan yararlanılarak daha verimli bir kaynak yönetimi sağlanır. Bu, geleneksel tek iş parçacıklı işleme yöntemine kıyasla işlem süresinde net bir iyileşme sağlar.

Kullanıcıya Açık Geri Bildirim Sağlanmalıdır:
Thread pool sayesinde işlemler daha hızlı tamamlandığı için, kullanıcıya işlem sonucuna dair anında geri bildirim verilir. Kullanıcı, işlemin sonucunu hızlıca almalıdır.

İş Gereksinimleri

Dosya Sıkıştırma ve Çıkarma İşlemleri:
Kullanıcı girdilerine dayalı olarak dosya sıkıştırma ve çıkarma işlemleri yapılacak ve thread pool teknolojisi ile paralel bir şekilde yönetilecektir. Bu, işlemlerin daha hızlı tamamlanmasını sağlar.

Fonksiyonel Gereksinimler

Multithreading Desteği ile Sıkıştırma ve Çıkarma İşlemleri:
Thread pool yöntemiyle, çok sayıda iş parçacığının aynı anda çalışması sağlanacak ve işlem süreleri kısaltılacaktır. Bu, geleneksel tek bir iş parçacığının kullanılmasından çok daha hızlıdır. Bu fonksiyonel gereksinim, performansı doğrudan etkileyen kritik bir özelliktir.

İşlem Tamamlandığında Kullanıcıya Bilgilendirme:
Her işlem tamamlandığında kullanıcıya bilgilendirme yapılacaktır. Thread pool, işlemler hızlı bir şekilde tamamlandığından, kullanıcıya gecikmesiz ve doğru geri bildirim sağlanacaktır.

Fonksiyonel Olmayan Gereksinimler

Performans:
Thread pool yöntemi ile işlem sürelerinin minimum seviyeye indirilmesi hedeflenmektedir. Multithreading, işlemleri paralel olarak işlediği için, büyük dosyaların yönetiminde önemli ölçüde hız sağlar. Bu, performans açısından önemli bir gereksinimdir.

Güvenilirlik:
İşlemler sırasında veri bütünlüğünün korunması çok önemlidir. Thread pool, iş parçacıklarının güvenli bir şekilde yönetilmesini sağlar ve veri bütünlüğünün korunmasına yardımcı olur. Ayrıca, thread pool sayesinde kaynak kullanımı optimize edilerek sistemin kararlı çalışması sağlanır.

Proje Planı

Kilometre Taşları

Multithreading yapısının başarıyla uygulanması: Thread pool yöntemi kullanılarak, işlem süreleri daha verimli bir şekilde yönetilecek ve hızlandırılacaktır. Bu, kaynakların verimli kullanılmasını sağlayarak paralel işleme yeteneğini artırır.

Dosya sıkıştırma ve çıkarma işlemlerinin doğruluğunun test edilmesi: Thread pool kullanarak yapılan paralel işlemler doğrulanacak, işlemlerin doğru ve hızlı bir şekilde tamamlanıp tamamlanmadığı test edilecektir.

Kaynak Tahsisi

Geliştirici: 1 kişi

Donanım: Thread pool destekleyen bir işlemci ve temel geliştirme araçları (IntelliJ IDEA)

Yazılım Mimarisi

Bu uygulama, **Thread Pool** yöntemini benimseyen bir yapı kullanır. Bu, çoklu iş parçacığı yönetimini verimli hale getirmek için kullanılacak ve her işlem için yeni iş parçacıkları yaratmak yerine, mevcut iş parçacıklarının bir havuzdan (thread pool) faydalanması sağlanacaktır. Bu sayede kaynak kullanımı optimize edilir ve işlem süreleri kısalır. Aşağıdaki katmanlar bu mimarinin bileşenleridir:

Sunum Katmanı (Presentation Layer)

Sunum katmanı, kullanıcı arayüzü yönetimini sağlar. Ancak, arka planda yapılan dosya sıkıştırma ve çıkarma işlemleri **Thread Pool** ile paralel hale getirilir. Kullanıcı, işlem sürecinin hızlı olduğunu hissedecek ve arayüz gecikmelerinden etkilenmeyecektir. Bu katman, JavaFX ile tasarlanmıştır ve dosya işlemlerinin hızlı bir şekilde gerçekleştirilmesini sağlar.

İş Mantığı Katmanı (Business Logic Layer)

İş mantığı katmanında, dosya sıkıştırma ve çıkarma işlemleri **Thread Pool** kullanılarak paralel hale getirilecektir. Bu katmandaki CompressController ve ExtractController sınıfları, her dosya işlemini bir iş parçacığına atamak yerine, bir thread pool'dan kaynak alacak ve işlemler paralel olarak yürütülecektir.

CompressController: Dosya sıkıştırma işlemi, mevcut thread pool iş parçacıkları tarafından paralel olarak yapılacaktır.

ExtractController: Dosya çıkarma işlemleri de thread pool kullanılarak daha verimli şekilde yönetilecektir.

Veri Katmanı (Data Layer)

Veri katmanında, dosya işlemleri sırasında verilerin geçici olarak saklanması sağlanır. Ancak, işlem paralel olarak gerçekleştirildiği için, **thread pool** sayesinde her bir iş parçacığı güvenli bir şekilde veri üzerinde çalışabilir. Böylece, işlem güvenliği ve veri bütünlüğü sağlanmış olur.

Kullanıcı Arayüzü (UI) Yapısı

Ana Ekran (Hello-view.fxml)

Kullanıcı, dosya sıkıştırma veya çıkarma işlemlerinden birini seçtiğinde, uygulama arka planda **thread pool** kullanarak işlemi hızla başlatacaktır. Bu ekran, kullanıcının seçimlerini alır ve yönlendirme sağlar.

Dosya Sıkıştırma Ekranı (Compress-view.fxml)

Sıkıştırma işlemi başlatıldığında, **thread pool** üzerinden işlem yönetimi sağlanır. Kullanıcı, ilerleme çubuğunu ve etiketleri görebilir, çünkü arka planda çoklu iş parçacıkları çalışmaktadır. İlerleme çubuğu, işlemlerin hızlı bir şekilde ilerlediğini görmesini sağlar.

Dosya Çıkarma Ekranı (Extract-view.fxml)

Dosya çıkarma işlemi de **thread pool** üzerinden paralel olarak yönetilir, böylece işlem süresi optimize edilir. Kullanıcı, çıkarma işlemi sırasında ilerlemeyi gerçek zamanlı olarak izleyebilir.

İş Akışları ve Kontrol Akışları

Ana Ekran (HelloController)

Kullanıcı, dosya sıkıştırma veya çıkarma seçeneklerinden birini seçtiğinde, ilgili ekrana yönlendirilir. Bu yönlendirme işlemi, çoklu iş parçacığından bağımsızdır.

Dosya Sıkıştırma Ekranı (CompressController)

compressFiles() fonksiyonu, dosyaları sıkıştırmak için thread pool'dan iş parçacığı alarak işlemi başlatır. Bu sayede işlemler paralel bir şekilde yürütülür. Her dosya sıkıştırıldıkça ilerleme durumu güncellenir.

Dosya Çıkarma Ekranı (ExtractController)

Kullanıcı, çıkarma işlemi başlattığında, extractFiles() fonksiyonu bir thread pool iş parçacığı tarafından yürütülür. Bu işlem paralel bir şekilde gerçekleştirilir.

Kullanılan Teknolojiler ve Kütüphaneler

JavaFX: Kullanıcı arayüzünü oluşturmak için kullanılır. Ancak, dosya işlemleri **ExecutorService** (Thread Pool) kullanılarak yürütülür. Java'nın ExecutorService sınıfı, iş parçacıklarını havuz içinde yönetmek için kullanılır ve kaynakları daha verimli kullanmak için tasarlanmıştır.

ZIP API: Dosya sıkıştırma ve çıkarma işlemleri için ZipOutputStream ve ZipEntry kullanılır, ancak bu işlemler **ExecutorService** tarafından yönetilen bir thread pool içinde paralel şekilde çalıştırılır.

Uygulama Fonksiyonel Yapısı

Dosya Seçimi

Kullanıcı dosya seçimi yaparken, işlemler doğrudan **thread pool**'a delegasyon yapmaz, ancak sıkıştırma ve çıkarma işlemleri **thread pool** tarafından hızlıca gerçekleştirilir.

Sıkıştırma/Çıkarma İşlemi

Seçilen dosyalar üzerinde işlem yapılırken, **thread pool** her dosya için farklı iş parçacıkları tahsis ederek işlem sürelerini kısaltır. Bu sayede, işlemler daha verimli bir şekilde tamamlanır.

İlerleme Takibi

Thread pool kullanımı sayesinde, işlemler daha hızlı gerçekleşeceği için ilerleme çubuğu daha hızlı güncellenir ve kullanıcı gerçek zamanlı olarak ilerlemeyi görür.

Hata Yönetimi

İşlem sırasında bir hata oluştuğunda, hata yönetimi **thread pool**'daki iş parçacıkları tarafından ele alınır, bu da hata yönetiminde hız sağlar.

Uygulama Tasarımı

Modülerlik

Her işlem için ayrı kontrolörler (controller) kullanılarak iş mantığı ve kullanıcı etkileşimleri ayrılmıştır. Bu tasarım, uygulamanın bakılabilirliğini artırır. **Thread pool**, her bir işlemi paralel hale getirerek her bir işlem için uygun iş parçacığı yönetimini sağlar.

Performans

Thread Pool kullanımı sayesinde dosya işlemleri paralel olarak gerçekleştirilecek ve işlem süreleri daha verimli hale gelecektir. Bu, kullanıcı deneyimini iyileştirecek ve uygulamanın performansını artıracaktır.

GELİŞTİRME SÜRECİ VE KOD BELGELERİ

Kod Yapısı ve Bileşenler

Bu proje, **JavaFX** kullanarak dosya sıkıştırma ve çıkarma işlemleri gerçekleştiren bir masaüstü uygulamasıdır. Uygulama, üç ana bileşenden oluşur:

HelloController (Ana Menü):

Kullanıcıya ana menü ekranını sunar ve kullanıcıdan dosya sıkıştırma veya çıkarma işlemi için seçim yapmalarını sağlar.

Ana menüde, "Dosya Sıkıştır" ve "Dosya Çıkar" butonları yer alır. Bu butonlar sırasıyla goToCompress() ve goToExtract() metodlarına bağlıdır.

CompressController (Dosya Sıkıştırma):

Kullanıcı, "Dosya Seç" butonuyla sıkıştırmak istediği dosya veya klasörleri seçer.

Seçilen dosyalar .zip formatında sıkıştırılır.

İlerleme durumu, bir **ProgressBar** ve etiketlerle kullanıcıya gösterilir.

Dosya sıkıştırma işlemi, çoklu iş parçacıkları (multithreading) kullanılarak yapılır ve işlemler **Thread Pool** yöntemi ile hızlandırılır.

ExtractController (Dosya Çıkarma):

Kullanıcı, sıkıştırılmış bir ZIP dosyasını seçer ve dosyayı çıkarır.

Seçilen dosya ve çıkarma işleminin durumu, etiketler ve **ProgressBar** ile kullanıcıya gösterilir.

Thread Pool Kullanımının Avantajları

Performans İyileştirmesi: Dosya sıkıştırma ve çıkarma işlemleri paralel olarak yürütülür, bu da işlem sürelerini önemli ölçüde kısaltır.

Verimli Kaynak Kullanımı: **Thread pool** sayesinde iş parçacıkları tekrar kullanılabilir, böylece her işlem için yeni bir iş parçacığı oluşturulmaz. Bu, kaynak kullanımını optimize eder.

Kontrol ve Yönetim: İş parçacıkları yönetimi daha kolay hale gelir. İşlem bitiminde shutdown() metodu çağrılarak tüm iş parçacıkları düzgün bir şekilde sonlandırılır.

Test Planı ve Test Senaryoları

Test Stratejileri

Testler, yazılımın doğru şekilde çalıştığını, kullanıcı etkileşimleri sırasında beklenen sonuçları verdiğini ve yazılımın hata vermeden stabil çalıştığını doğrulamak amacıyla gerçekleştirilmiştir. Testler, aşağıdaki kategorilere ayrılmıştır:

Fonksiyonel Testler: Kullanıcı etkileşimlerinin beklenen sonuçları verip vermediğini kontrol eder. Her dosya sıkıştırma ve çıkarma işlemi, ilgili kontrolörler aracılığıyla thread pool yöntemini doğru şekilde kullanarak yapılır.

Kullanıcı Arayüzü Testleri: Butonlar, etiketler ve ProgressBar gibi öğelerin doğru şekilde göründüğü ve işlevsel olduğu doğrulanır. Kullanıcı, her işlemde ilerleme durumunu izleyebilir.

Performans Testleri: Çoklu dosya sıkıştırma ve çıkarma işlemlerinin hızlı ve verimli bir şekilde gerçekleşip gerçekleşmediği test edilir. Thread pool kullanımı sayesinde, her dosya için yeni bir iş parçacığı başlatmak yerine mevcut iş parçacıkları paralel şekilde kullanılır, böylece kaynak kullanımı optimize edilir.

Test Senaryoları

Ana Menüden Dosya Sıkıştırma Ekranına Geçiş:

Test Adımı: "Dosya Sıkıştır" butonuna tıklanır ve sıkıştırma ekranına geçiş yapılır.

Beklenen Sonuç: CompressController ekranı yüklenir ve kullanıcı dosya seçimi yapabileceği bir arayüze yönlendirilir.

Dosya Seçimi ve Sıkıştırma İşlemi:

Test Adımı: "Dosya Seç" butonuna tıklanarak bir dosya seçilir. Ardından "Sıkıştır" butonuna tıklanır ve seçilen dosya sıkıştırılır.

Beklenen Sonuç: Dosya başarıyla sıkıştırılır ve ilerleme durumu kullanıcıya doğru şekilde gösterilir. Thread pool üzerinden her dosya için farklı iş parçacıkları tahsis edilir.

Dosya Çıkarma İşlemi:

Test Adımı: "ZIP Dosyası Seç" butonuna tıklanarak bir ZIP dosyası seçilir. "Çıkar" butonuna tıklanır ve dosyalar çıkarılır.

Beklenen Sonuç: Dosya çıkarma işlemi başarıyla tamamlanır ve ilerleme durumu güncellenir. Thread pool kullanımı sayesinde, çıkarma işlemi paralel şekilde yönetilir ve işlem süresi kısalır.

Yanıt Verme ve Hata Durumları:

Test Adımı: "Sıkıştır" işlemi sırasında, hiçbir dosya seçilmediğinde hata mesajı gösterilir.

Beklenen Sonuç: Uygulama, hata mesajı olarak "Lütfen en az bir klasör seçin!" mesajını gösterir. Hata durumları kullanıcı dostu şekilde ele alınır ve kullanıcıyı bilgilendirir.

Gerçekleştirilen Testlerin Sonuçları

Başarıyla Geçen Testler:

Tüm fonksiyonel testler başarılı bir şekilde tamamlanmıştır. Kullanıcı etkileşimleri doğru şekilde çalışmaktadır. Dosya sıkıştırma ve çıkarma işlemleri thread pool yöntemi ile verimli bir şekilde gerçekleştirilmektedir.

Performans Testleri:

Dosya sıkıştırma ve çıkarma işlemleri hızlı bir şekilde gerçekleştirilmiştir. Çoklu iş parçacıkları doğru şekilde yönetilmektedir ve işlem süreleri önemli ölçüde kısalmıştır. Thread pool kullanımı sayesinde, her dosya için yeni bir iş parçacığı başlatmak yerine mevcut iş parçacıkları paralel bir şekilde kullanılmakta ve kaynaklar verimli bir şekilde yönetilmektedir.

Hata Durumu:

Hata mesajları doğru bir şekilde gösterilmektedir ve kullanıcı dostudur. Herhangi bir hata durumunda, kullanıcıya uygun bir hata mesajı ile bilgilendirilme yapılır. Bu, kullanıcı deneyimini olumlu şekilde etkilemektedir.

Kullanıcı Kılavuzu

Yazılımın Kullanımı

Bu yazılım, kullanıcıların klasörleri sıkıştırmalarına ve çıkarılmasına olanak tanır. Kullanıcı, aşağıdaki adımları izleyerek işlemleri gerçekleştirebilir:

Ana Menü:

"Dosya Sıkıştır" butonuna tıklayarak dosya sıkıştırma ekranına geçiş yapabilirsiniz.

"Dosya Çıkar" butonuna tıklayarak dosya çıkarma ekranına geçiş yapabilirsiniz.

Dosya Sıkıştırma:

"Dosya Seç" butonuna tıklayarak bir dosya veya klasör seçin.

"Sıkıştır" butonuna tıklayarak seçilen dosyaları sıkıştırın.

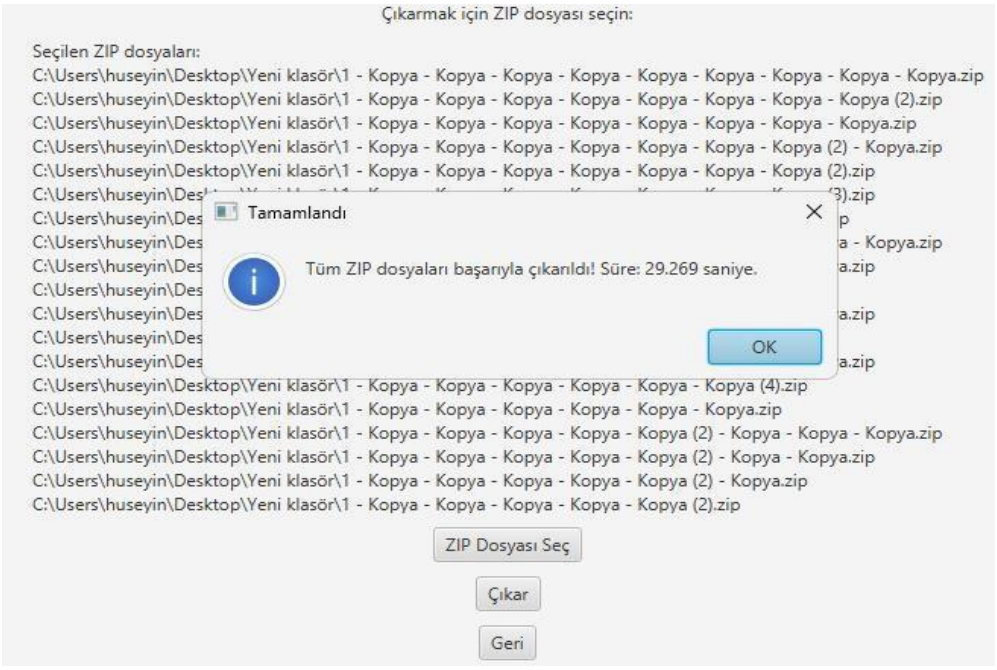
İlerleme durumu bir ProgressBar ve etiketlerle gösterilecektir.

Dosya Çıkarma:

"ZIP Dosyası Seç" butonuna tıklayarak bir ZIP dosyası seçin.

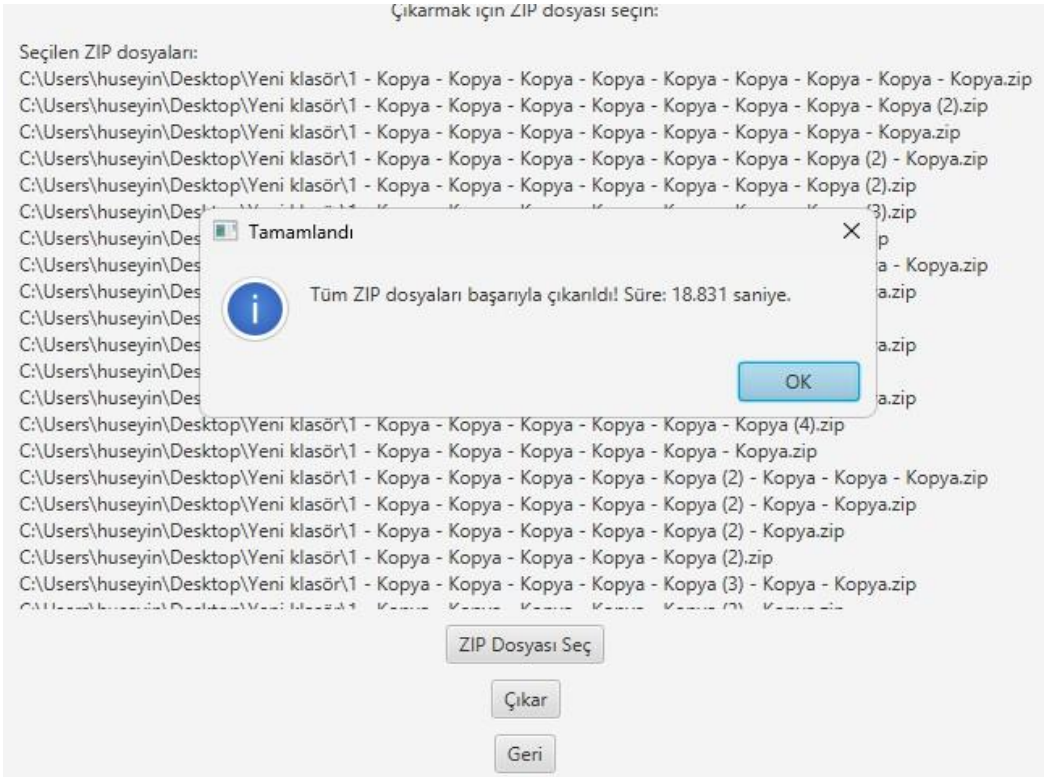
"Çıkar" butonuna tıklayarak dosyayı çıkarın.

İlerleme durumu bir ProgressBar ve etiketlerle gösterilecektir.



Aynı boyutta 50 adet ZIP dosyası kullanıldı.

Manuel thread yönetimi daha fazla kaynak tüketebilir ve daha karmaşık hata yönetimi gerektirir.



Aynı boyutta 50 adet ZIP dosyası kullanıldı.

Verimli thread yönetimi, daha az sistem kaynağı kullanımı sağlayan bir yaklaşımdır, çünkü thread havuzunu kullanır.

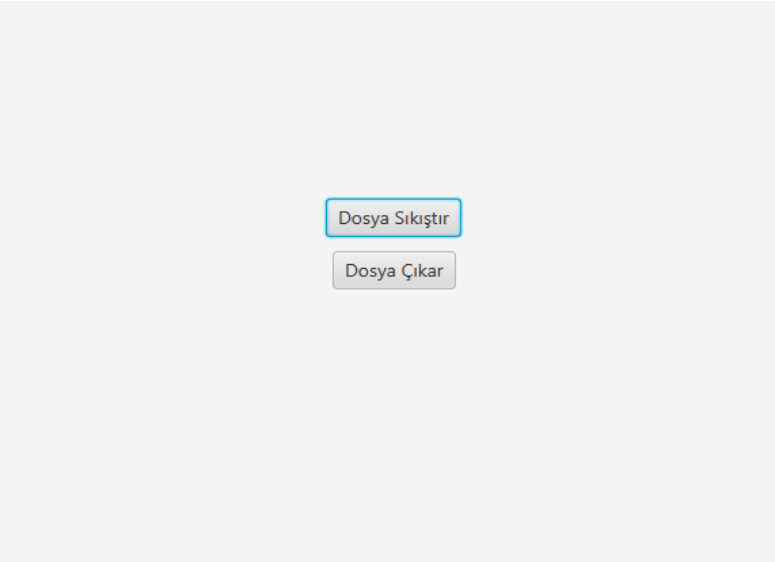
Thread Pool Yöntemi : Bu yöntemde, iş parçacıkları bir havuzda (thread pool) yönetilir. Havuz, önceden belirlenen sayıda iş parçacığına sahiptir ve bu iş parçacıkları, iş yapılması gerektiğinde yeniden kullanılır. Yeni bir iş parçacığı yaratmak yerine, mevcut iş parçacıkları tekrar kullanılarak işleme alınır. Bu yöntem, kaynakları daha verimli kullanır çünkü her iş için yeni bir iş parçacığı oluşturulmaz, böylece sistemin kaynakları (özellikle işlemci ve bellek) daha etkili bir şekilde paylaşılır. Ayrıca, fazla sayıda işlem olduğunda, iş parçacığı sayısı sınırlı tutulur, bu da sistemin aşırı yüklenmesini engeller ve verimliliği artırır.

Her Dosya İçin Yeni Thread Başlatma : Bu yöntemde, her zip dosyası için yeni bir iş parçacığı başlatılır. Dosya sayısı arttıkça, her dosya için ayrı ayrı iş parçacıkları yaratılır. Bu, fazla sayıda iş parçacığı oluşturulmasına yol açar ve her iş parçacığının başlatılması, yönetilmesi ve sonlandırılması işlemci kaynaklarını tüketir. Sistem, bu kadar çok sayıda iş parçacığını yönetmekte zorlanabilir, bu da kaynak tüketimini artırır ve işlemci limitlerine ulaşılmasına neden olabilir. Ayrıca, iş parçacıklarının yönetimi daha manuel olduğu için, sistemde fazla iş parçacığı olduğunda verimlilik kaybı yaşanabilir.

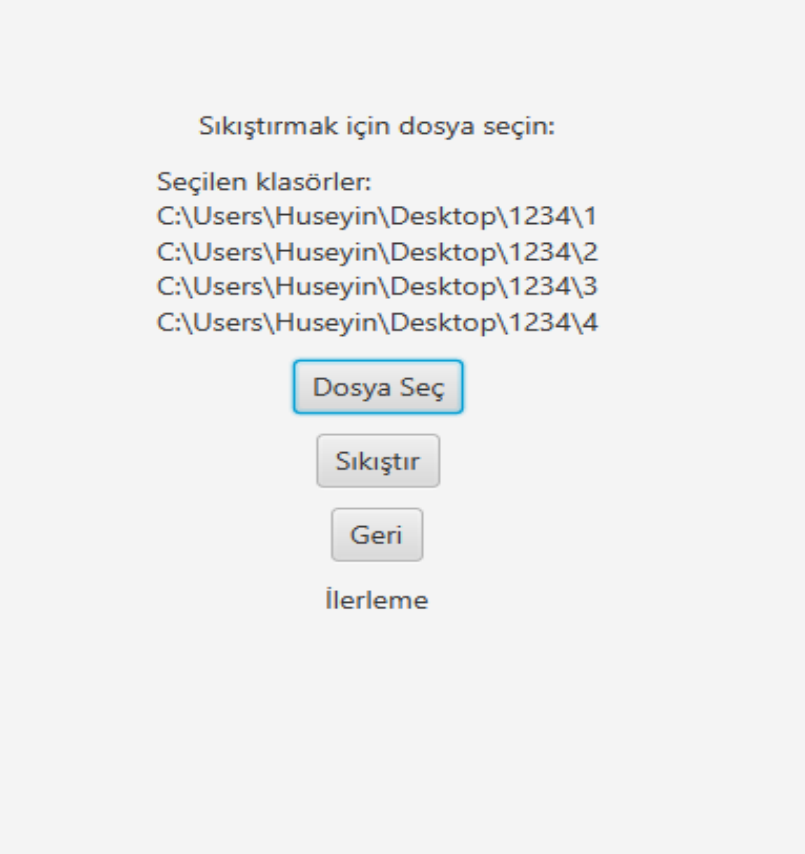
Sonuç olarak, Thread Pool Yöntemi, iş parçacığı yönetimini daha optimize bir şekilde yaparak kaynakları verimli kullanır ve sistemin performansını artırır. Her Dosya İçin Yeni Thread Başlatma Yöntemi ise, her işlem için yeni iş parçacıkları oluşturduğu için kaynakları gereksiz yere tüketir ve verimliliği düşürür.

Arayüz Açıklamaları

Ana Menü: Kullanıcı, burada dosya sıkıştırma veya çıkarma işlemini seçebilir.



Dosya Seçme ve İlerleme Durumu: Kullanıcı, sıkıştırma veya çıkarma işlemlerinin ilerleyişini ProgressBar ve etiketlerle görebilir.



Geri Dön: Kullanıcı, işlemi iptal edip ana menüye geri dönebilir.

Örnek Kullanım Senaryoları

Dosya Sıkıştırma: Kullanıcı, klasörleri seçer, ardından "Sıkıştır" butonuna tıklayarak seçilen dosyaları ZIP formatında sıkıştırır.

Dosya Çıkarma: Kullanıcı, ZIP dosyalarını seçer, ardından "Çıkar" butonuna tıklayarak dosyaları çıkarır.

Kaynak Kod: https://github.com/huseyincaglar1/DosyaArsivleyici