**THREAD KONU ANLATIMI**

**15.07.2025**

## Executor Nedir?

Thread Oluşturmanın farklı yöntemleri vardır

new Thread(...) ile thread'i manuel olarak başlatırsın.

Örnek:

```
Thread t = new Thread(() -> {
    // İş yapılır
});
t.start();
```

Bu yöntem:

- Basittir ama yetersizdir.
- Büyük uygulamalarda thread yönetimini zorlaştırır.
- Thread'ler sürekli yaratılır ve sistem yükü artar.
- Thread havuzu (pool) yoktur.

Bunun yerine Executor kullanılabilir

Executor, Java'nın **modern ve daha profesyonel** çoklu iş parçacığı yapısıdır.

Temel fikir: "Thread'leri elle oluşturma. Ben senin yerine yönetirim."

Thread'in çalıştırmasını istediğin yere Runnable veya Callable<T> verirsin.

**Runnable ->** void (yani hiçbir şey dönmez) – Exception atamaz (try-catch içinde yazılmalı)

**Callable<T>** -> T tipinde bir değer döner (generic) - Exception Atabilir (örneğin IOException)

**Örnek:**

```
public Callable<FileStats> createFileAnalysisTask(Path filePath) {
        return () -> {
            try {
                logger.info("Dosya analizi başlatılıyor: {} - Thread: {}",
                        filePath.getFileName(), Thread.currentThread().getName());

                //Sureyi baslatiriz

                // Dosya analizi
                FileStats stats = analyzeFile(filePath); // bu metodun satir ve sutun hesapladigi varsayilir

                //Sureyi bitiririz

                logger.info("Dosya analizi tamamlandı: {} - Süre: {} ns - Thread: {}",
                        filePath.getFileName(), processingTime, Thread.currentThread().getName());

                return stats;

            } catch (Exception e) {

            }
        };
    }
```

Burada bizim dün gece düşündüğümüz FileStats entity'si geriye döndüreceği için Runnable yerine Callable Kullandık. Daha sonra bu metot bir Thread tarafından yürütülebilir task haline gelir

**Peki Şimdi Nasıl new thread() demeden Executor ile bu metodu çalıştıracağız?**

İlk olarak Executor oluşturmak gerek bunun için bir config dosyası yazacağız.

Neden böyle yapıyoruz? Çünkü analiz işlemi için bir thread havuzu arşivleme için ayrı thread kullanacağız ve bu havuzların kendine göre kuralları var örneğin max 10 thread. Bu yüzden bunları bir config dosyasında tanımlayıp sonrasında ilgili Executor'u çağırırsak daha iyi olur.

**Örnek**

```
@Configuration
public class ThreadPoolConfig {
    /**
     * Dosya analizi için thread havuzu
     * Her dosya için ayrı thread kullanılacak (maksimum 10 thread)
     */
    @Bean(name = "fileAnalysisExecutor")
    public Executor fileAnalysisExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(2);                    // Minimum thread sayısı
        executor.setMaxPoolSize(10);                    // Maksimum thread sayısı (proje gereksinimi)
        executor.setQueueCapacity(20);                  // Bekleyen işler için kuyruk
        executor.setThreadNamePrefix("FileAnalysis-");  // Thread isim öneki
        executor.setKeepAliveSeconds(60);               // Boşta kalan thread'lerin yaşam süresi
        executor.initialize();
        return executor;
    }

    /**
     * Arşivleme işlemi için thread havuzu
     * Tek thread kullanılacak (analiz bittikten sonra çalışacak)
     */
    @Bean(name = "archiveExecutor")
    public Executor archiveExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(1);                    // Tek thread
        executor.setMaxPoolSize(1);                     // Maksimum 1 thread
        executor.setQueueCapacity(10);                  // Bekleyen işler için kuyruk
        executor.setThreadNamePrefix("Archive-");       // Thread isim öneki
        executor.setKeepAliveSeconds(60);
        executor.initialize();
        return executor;
    }
}
```

Yukarıdaki örnekte analiz ve arşivleme için 2 thread havuzu oluşturduk ve yönetimi manuel (new thread()) yapmak yerine daha esnek ve basitleştirdik.

### Peki bu Executorları Nasıl Kullanacağız?

```
//Executorlardan Nesne olusturuyoruz
@Autowired
@Qualifier("fileAnalysisExecutor") //Bu anotasyon asagidaki nesneyi Config dosyasinda "fileAnalysisExecutor" ismini verdigimiz executora esitler
private Executor fileAnalysisExecutor;

@Autowired
@Qualifier("archiveExecutor") //Yukaridakiyle ayni mantikta calisir
private Executor archiveExecutor;

List<Future<FileStats>> futures = filePaths.stream()
    .map(filePath -> fileAnalysisExecutor.submit(
        fileProcessingService.createFileAnalysisTask(filePath)
    ))
```