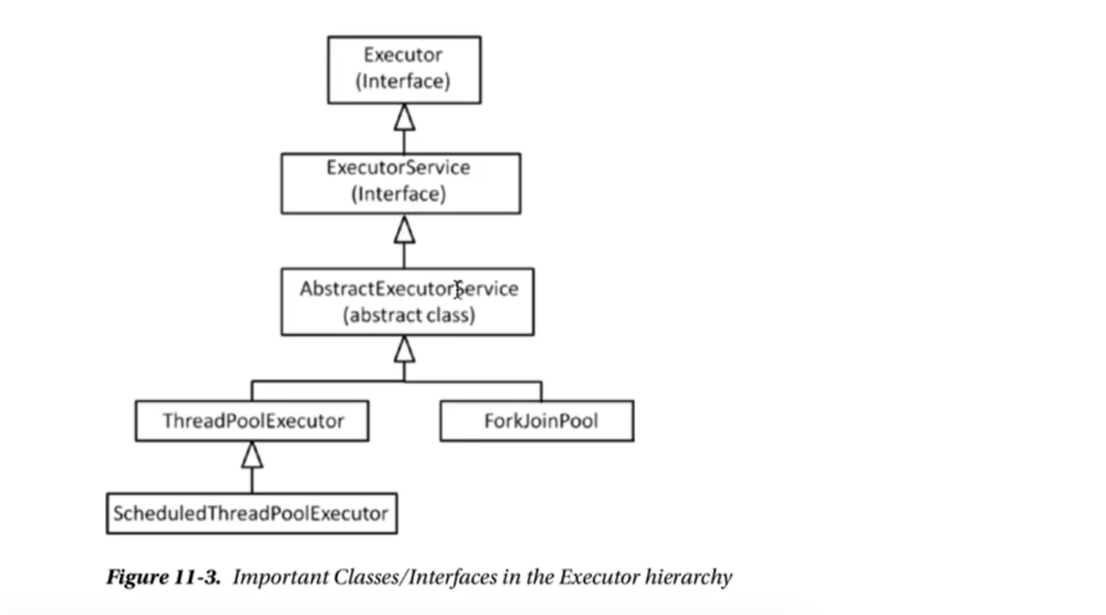
**EXECUTOR SERVICE**



Java8 ile birlikte Thread nesnelerinin new()’lenmeden oluşturulması ve yönetilmesi mevcuttur. Eğer thread’lerle çalışılması gerekiyorsa ExecuterService üzerinde çalışılması önerilmekte. Ancak Java processi yeni kodda hiçbir zaman durmayacaktır. Yani executor service arka planda çalışmaya ve task alıp işlemeyi beklemektedir. Açık olarak executor servisi stop edebiliriz. Executor interfacesini kalıtmakta ve içerisinde farklı methodları mevcuttur

service.shutdown(); // executor üzerinde bir task var ise onu bitirmesini bekler.

service.shutDownNow(); // executor üzerinde bir taskın olup olmadığna bakmaksızın stop eder.

service.awaitTermination(1, TimeUnit.SECONDS) // executor'ın 5 sn sonra kapanmasını sağlar..

callableExample() methodunda bu konuda örnek yapılmıştır.

callableExampleTask() methodunda bu konuda örnek yapılmıştır.

**Runnable ve Callable Arasındaki Fark**

Arasındaki fark Runnable void geriye bir method döndürmekte, Callable ise generic olarak verilen type döndürülmektedir. Birde checked exception fırlatmaktadır.

differenceBetweenRunnableAndCallable() methodunda bu konuda örnek yapılmıştır.

Scheduling Task

* Planlanmış görevlerdir.
* Tanımlı görevleri belirtilen süre sonra çalışmasını sağlar.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

**schedule**() : Belirtilen gecikmeden sonra bir görev oluşturur ve yürütür.

Overload versiyonları mevcuttur.

scheduledTaskExample() methodunda açıklanmıştır.

Method içerisinde tasklar belirledik. Daha sonra schedule methoduyla bu taskların ne zaman çalışacağını belirttik.

**scheduleAtFixedRate**(): Belirtilen gecikmeden sonra bir görev oluşturur ve yürütür. Daha sonra tanımlı zamanda yeniden çalışır.

scheduledAtFixedExample() methodunda açıklanmıştır.

Tanımlı görev 3 saniye sonra çalışacak ve her 5 saniyede bir yeniden çalışacak.

**scheduleWithFixedDelay**(): Belirtilen gecikmeden sonra bir görev oluşturur ve yürütür. Daha sonra tanımlı zamanda yeniden çalışır.

scheduledWithDelayExample() methodunda açıklanmıştır.

Tanımlı görev 3 saniye sonra çalışacak ve her 5 saniyede bir yeniden çalışacak.

İki method arasındaki fark **scheduleAtFixedRate** kendinden önce bir task varsa onun bitip bitmesine bakmadan belirtilen period boyunca çalışır. **scheduleWithFixedDelay** ise kendinden önceki taskın bitmesini bekler.

POOL

Singlethread kullandığımızda bir executer service birde fazla task verdiğimizde bir birini beklemek durumundaydı. Burada ise paralel bir şekilde biden fazla thread’in çalışmasını sağlamaktadır.

threadPoolExample() methodunda 4 thread olusturulup parallel şekilde çalışması sağlanmıştır.

CYCLİCBARRİER

Takımların ikişerli olduğu dört kişiyle oynanan bir oyun düşünelim. Bunun için bize dört adet thread gerekli. Takımların dört kişilik olduğunu belittikten sonra aslında bir bariyer(bekleme noktası) tanımlamış oluyoruz. Dörde tamamlandığında çalışmaya devam edecek.

FORK / JOIN FRAMEWORK

İşleri parçalara bölme mantığı vardır. Work-stealing algoritmasına göre çalışır yani bir çalışan işini bitirdiği zaman diğerine geçer. Genel olarak işleri ayır taskları birleştir ve sonuçları oluştur şeklinde bir mantığ ıvardır.

SumOfNUsingForkJoin örneği:

Forjoinpool oluşturuyoruz ve 10 thread ile çalışmasını belirtiyoruz.

Ondan sonra invoke (çağırmak) ediyoruz. İlgili taskı buraya veriyoruz paramet eolarak. O ve N şeklinde nerden baslayacak nereye gidecek şeklinde

PARALEL STREAM