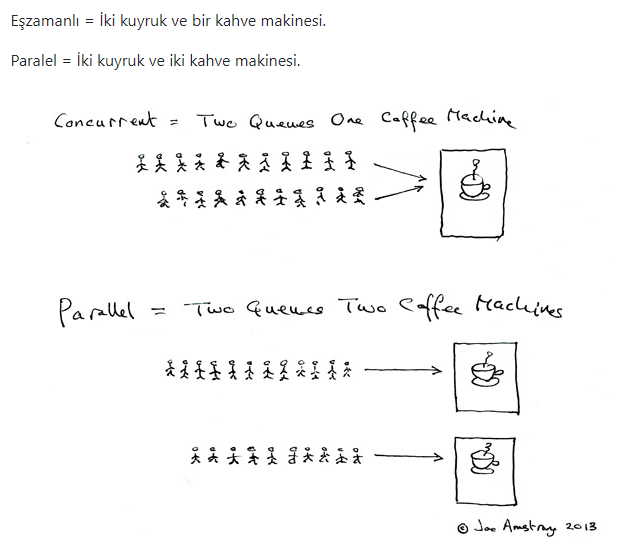
# 1. Concurrent programlama ve Parallel Programlama nedir ? Aralarında çalışma şekli olarak nasıl bir fark bulunmaktadır ?

**Paralel programlama (Parallelism)**

Bir görevin paralel olarak işlenebilecek daha küçük alt görevlere ayrılarak birden fazla CPU’da, threadler vasıtası ile, paralel olarak yürütülmesi anlamına gelir. Paralel işlemler gereken veri ve kaynak bakımından biribirinden bağımsızdır ve çoğunlukla senkronizasyona ihtiyaç duymazlar.



# 2. Mutex ve Semaphore kavramlarını açıklayınız. Hangi tür durumlarda bunlara başvurmamız gerekir ?

**Semaforlar**

Semaforlar process ve thread sekronizasyonu için kullanılır. Mutex’ler ikilik semaforlardır. Mutex’ler aynı proses ve thread’leri tarafından kullanılır. Fakat semaforlar farklı prosesler tarafından da kullanılabilir.

Semaforların pthread mutex'lerden temel farkı, sadece aynı uygulamanın thread'leri arasında değil, farklı process'ler arasındaki senkronizasyon işlemleri için de kullanılabilmesidir.

Aynı uygulama içerisinde semafor veya mutex kullanımı açısından performans farkı görülmese de, mutex kullanımı tercih edilmelidir. Mutex kullanımı sayesinde, ancak mutex'i lock etmiş thread'in tekrar unlock yapabilmesini garanti etmiş oluruz.

Semafor kullanılması durumunda, bir thread tarafından artırılan semafor değeri, başka bir thread tarafından düşürülebilir. Bu nedenle semaforlar, eşzamanlı uygulama geliştirme sürecindeki goto deyimi olarak da nitelendirilir.

Bununla birlikte, farklı process'ler arasındaki senkronizasyon probleminin çözümü ve async-signal safe yapısı nedeniyle multi-threaded uygulamalarda signal handler içerisinden kullanılabilmesi, semafor kullanımının avantajlı olduğu noktalardır

# 3. Java’da Error ve Exception arasındaki fark nedir ? Throwable ile ilişkileri nasıldır ? Hangi tür durumlarda Error hangi tür durumlarda Exception meydana gelebilir ? Örneklerler açıklayınız.

Javada tüm hatalar sınıflar tarafından temsil edilir, yani bir hata oluştuğunda bir hata objesi oluşturulur. Java dilinde tüm hataların atası Throwable sınıfıdır. Throwable sınıfının iki tane doğrudan alt sınıfı bulunur, Exception ve Error. Error sınıfı doğrudan bizim kodumuzla ilgili olmayan JVM ile ilgili hatalar için kullanılır ve bu hatalar kontrolümüz dışındadır.

Exception sınıfı ise doğrudan yazdığımız kodla ilgili bizim kontrolümüzde olan hatalar için kullanılır.

# 4. Spring’te yer alan @Scheduled anotasyonunun kullanım amaçlarını ve kullanım şeklini açıklayınız.

Spring Framework bize anotasyonları kullanarak, neredeyse hiç threadler için kod yazmadan tasklarımızı yazmamızı sağlar. Yani bizi daha da soyutlar, altta yer alan detaylar ile uğraştırmaz. Periyodik olarak yapmak istediğimiz metodun başına @ Scheduled anotasyonunu eklememiz yeterli olur.

Uygulamamıza @ EnableScheduling anotasyonunu ekleriz. Zamanlamak istediğimiz metoda da @ Scheduled anotasyonunu ekleriz. Burada 3 tip Schedule bulunmakta. Bunlar:

fixedDelay : Sabit gecikme süresi ile, taskımız bir önceki taskın bitiminden sonra belirlediğimiz süre kadar bekler. Yani ‘fixedDelay=5000’ dediğimizde önceki taskın bitiminden 5 saniye sonra yeni task başlar. Tasklar arasında 5 saniye olduğundan emin oluruz.

fixedRate: Sabit belirlenen süre ile, taskın gerçekleşeceği zaman bir önceki taskın başladığı zamandan hesaplanır. Yani ‘fixedRate=5000’ dediğimizde önceki task başladıktan sonra 5 saniye içerisinde bu task gerçekleşecek demektir.

cron: Aynı Linux’da bulunan cron yapısını burada da kullanabiliriz. Basit bir cron expression ile işlemlerimizi istediğimiz belli bir periyoda zamanlayabiliriz.

# 5. Spring’te yer alan @Async anotasyonunun kullanım amaçlarını ve kullanım şeklini açıklayınız.

Bir bean’in methoduna @Async annotationu eklemek onun main thread’den farklı olarak ayrı bir threadde çalıştırılmasını sağlar.Yani call edilen methodun tamamlanmasını call eden kısım beklemez. Kod böylece async olarak çalışmış olur.

@Async annotationu ile Async bir method yaratmak çok kolay.

2 Adımda(3. Adım opsiyonel) nasıl yapabileceğinizi gösterip, örneklere geçelim.

**1 Enable Async Support**

Asynchronous process’leri, Java configuration’ı ile enable etmeliyiz(@EnableAsync). @EnableAsync annotationu @Async annotationuna sahip methodları arayacak ve bu methodları background thread pool’larında çalışacaktır.

|  |
| --- |
| @SpringBootApplication |
|  | @EnableAsync |
|  | public class SpringAsyncApplication { |
|  |  |
|  | public static void main(String[] args) { |
|  | SpringApplication.run(SpringAsyncApplication.class, args); |
|  | } |
|  | } |

2. @Async annotationu Async yapmak istediğimiz method üzerine eklemek

|  |
| --- |
| @Async |
|  | public void asyncMethod() { |
|  | System.out.println("Async Method"); |
|  | } |

3. İsteğe bağlı olarak Executor’ın ayarlarını özelleştirebilirsiniz.

|  |
| --- |
| @Bean(name = "asyncExecutor") |
|  | public Executor asyncExecutor() |
|  | { |
|  | ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor(); |
|  | executor.setCorePoolSize(3); |
|  | executor.setMaxPoolSize(3); |
|  | executor.setQueueCapacity(100); |
|  | executor.setThreadNamePrefix("AsynchThread-"); |
|  | executor.initialize(); |
|  | return executor; |
|  | } |

@Async annotationu kullanırken dikkat edilmesi gereken 2 kural vardır.

1. Method public olmalıdır.

2. Aynı class içerisinden @Async bir methodu çağırırsanız çağrılan method asynchronous olarak çalışmaz.

# 6. High Availability (HA) kavramını kısa açıklayınız.

Network cihazlarında , Sunucu sistemlerinde vb bu kavram sıklıkla kullanılır. Buradaki amaç kullanmış olduğumuz sistem ne olursa olsun, onun sürekliliğini sağlamaktır. Yani Yüksek Erişilebilirlik demek sunulan servisin herhangi bir nedenle kesintiye uğramaması için birden çok makina veya cihaz ile desteklenmesi şeklinde özetlenebilir. Örneğin mail server olarak Exchange Server kullanıyorsanız ve siz mail serverım HA özelliğine sahip olsun derseniz aslında amacınız, sunuculardan birine veya storage’ a veya elektirik sorunlarına veya network sorunlarına karşı dayanıklı bir yapı oluşturmak istiyorsunuz demektir. Bu nedenle yedekli elektirik alt yapısı, birden çok ups, exchange server için cluster bir mimari ( CCR,SCR,SCC veya DAG vb ), network için yedekli switchler vb yapılar kurarak siz mail altyapınızı High Available yapmış olursunuz.

# 7. Entity ve Value Object kavramlarını Domain Driven Design (DDD) kapsamında açıklayınız.

Kendine ait unique bir kimliği olan nesneler Entity olarak adlandırılırken, kendine ait bir kimliği olmayanlar ise value object olarak adlandırılıyor.

Entityler domainimiz içerisindeki bir kavrama denk gelirler. DDD nesne yönelimli programlamaya sımsıkı bağlıdır. İş mantığımızı servislere değil bütünüyle entitylere yığmamızı öneriyor. Kısaca sınıflarımızı, veritabanımızdaki bir tabloya karşılık gelecek şekilde değil, iş mantığımızı oturtacağımız yapıda, bir başka deyişle entitylere logic ekleyerek oluşturmalıyız.

# 8. Ubiquitous Language kavramını DDD kapsamında açıklayınız. Sizce neden önemli olabileceğini belirtiniz.

Oldukça zor telaffuz edilen bu kavram, DDD’ nin temel taşlarından birini oluşturur. Biz yazılımcıların istenilen çıktıyı üretebilmemiz ve bu çıktının sürekliliğini sağlayabilmemiz için Domain Expert’ ler ile aynı dili konuşacak seviyeye gelmemiz gerekiyor. Sonrasında da edindiğimiz bu tecrübeyi uygulamalarımızı geliştirirken kullandığımız method ve sınıflara expertin kullandığı kavramların isimlerini vererek aktarmamız gerekiyor. Projemizde kullanacağımız her servisin domainde bir karşılığı olması gerekiyor. Böylelikle projede yer alan herkes bu ortak dili konuşabilir birbirini anlayabilir oluyor.

# 9. Core Domain, Supporting Domain, Generic Domain kavramlarını DDD kapsamında açıklayınız.

**Core domain** : iş için gerekli olan en önemli alt etki alanı. Onsuz, iş başarısız olur. Uygulamak için ilk çözümü seçmeniz gerekirse - temel alanla başlayın.

**Supporting subdomain** : iş için Çekirdek etki alanından daha az değerli olan alt etki alanı. Onsuz iş, bir süre hayatta kalabilir bile. Ancak yine de oldukça önemlidir (çekirdek alanı destekler), ayrıca alana özgüdür ve geliştirilmesi gerekir. Bu durumda, herhangi bir nedenle, sorunu çözmek için mevcut bir yazılımı veya bileşeni satın alamıyoruz

**Generic subdomain :** İşletmeler için Core alan adından daha az değerli olan alt alan. Ayrıca, raftan satın alınmasına izin verecek kadar geneldir (destekleyici etki alanının aksine).

# 10. Anemic Domain Model ve Rich Domain Model kavramlarını kıyaslayarak açıklayınız.

Anemic domain model, hiç bir fonksiyonelliği olmayan içerisinde hiç bir fonksiyon barındırmayan sadece propertylerden oluşmuş bir modeldir ve düşünüldüğünde OOP’e aykırı bir yapı teşkil eder.

Anemik Etki Alanı Modellerinin ana dezavantajlarını ele almanın iyi bir yolu, Zengin Etki Alanı Modeli uygulamaktır. Anemik Etki Alanı Modeliyle arasındaki temel fark, etki alanı mantığımızın etki alanı varlıklarımızın, veri ve davranışımızın bir parçası olmasıdır. Bu mantık, varlığın nasıl örneklendirildiğini, doğrulandığını ve işletildiğini yönlendirir ve kontrol eder ve müşterinin tutarsız bir duruma sahip varlıklara sahip olmasını önler.