1) **Concurrent programlama ve Parallel Programlama nedir? Aralarında çalışma şekli olarak nasıl bir fark bulunmaktadır?**
   Concurrent programlamada birden fazla görevi aynı işlemcide ilerletme işi olarak tanımlayabiliriz. Paralel programlamada ise birden fazla CPU ile birden fazla işi aynı anda ilerletme olarak tanımlayabiliriz.
   Concurrent programlama için tek işlemci yeterken paralel programlada birden çok işlemciye ihttiyacımız vardır.

2) **Mutex ve Semaphore kavramlarını açıklayınız. Hangi tür durumlarda bunlara başvurmamız gerekir?**
   Mutex paylaşılan kaynaga erişimi düzenlemek için kullanılan basit veri yapılarıdır. Paylaşılan kaynağa erişilen bölgeye Critical Section olarak adlandırılır.Kaynakla işi olan Thread, Mutex'in sahipligini almaya çalışır. Eğer başka bir Thread kullanıyorsa diger Thread beklemeye alınır. Semaphore temel olarak çok sayıda bulunan resourceların birçok kullanıcı tarafından sırayla kullanılmasını sağlar. Semaphore'lar aralarında sinyalleşme gereken Producer-Consumer senaryolarında sinyalleşme açısından bir çözüm sunmaktadır. Semaphore'un uyutulup Thread'in bütün stateini tutmak maliyetli olacagı için bazen Thread uyutulmaz ve Busy Waiting işlemi ile canlı tutulur. Bu işleme **Spinlock Semaphore** adı verilir

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.**
   Error uygulamadaki yakalayamaması gereken ciddi sorunları belirtirken, Exception yakalamak istediğimiz istisnai durumları belirtir. Her ikiside throwable dan extend eder. Dosyanın bulunamaması gibi veya parse edilememesi gibi durumlar exceptiondır ve bu tür hataları yakalamaya çalışırız. Fakat Ram in bitmesi gibi kritik durumlar Error'lardır ve bu tür durumları yakalayamayız.

4) **Spring'te yer alan @Scheduled anotasyonunun kullanım amaçlarını ve kullanım şeklini açıklayınız.**
   @Scheduled anotasyonu threadler ile kod yazmamızı kolaylaştırıp arkadaki işleri soyutlayan bir anotasyondur. Periyodik olarak yapmak istediğimiz metodun başında kullanmamız yeterlidir. 3 tip Schedule bulunmaktadır. fixedDelay sabit gecikme süresi önceki task bitiminden itibaren çalışır, fidexRate sabit belirlenen süre ile önceki task başlangıcından itibaren çalışır. Cron ise sürekli olarak yapılmasını istedigimiz işleri planlarken kullanırız linuxdeki cron ile aynıdır.

5) **Spring'te yer alan @Async anotasyonunun kullanım amaçlarını ve kullanım şeklini açıklayınız.**
   Bir bean'e Async anotasyonunu eklemek onu main threadden farklı olarak bir threadde çalıştırmamızı sağlar.

6) **High Availability (HA) kavramını kısa açıklayınız.**
   High Availability, bir sistemin ya da sistem bileşenlerinin istenen operasyonel süre içerisinde hizmet vermeyi devam ettirme yeteneğidir. Yüksek erişilebilirlik, sistemin bir kullanıcının yaptığı isteğe yanıt vermesi için gereken süreyi tanımlamak için kullanılır.

7) **Entity ve Value Object kavramlarını Domain Driven Design (DDD) kapsamında açıklayınız.**
Entity kendini diğer nesnelere nazaran tekilleştirebilmek için bir kimliğe(Id) sahip olan nesnelerdir. Entity, özünde Entity Framework'den aşina olunduğu gibi yeryüzündeki herhangi bir şey için modellenmiş nesnelere karşılık gelmektedir. Bahsedilen kimlik ise bu nesnelerin her biri için yaratıldığı süreçten itibaren diğerlerinden ayırmamızı sağlayan ve değişmeden taşınan Id değeridir.
Value object, herhangi bir kimlik(Id) değeri olmayan ve böylece aynı değerlere sahip iki value object nesnesinin değersel açıdan aynı olarak kabul edilebilir olmasını sağlayan ve dolayısıyla birbirlerinin yerine geçebilecekleri anlamına gelen bir nesnedir. İşte bu nedenle value object'ler her daim değişmez(immutable)dirler

8) **Ubiquitous Language kavramını DDD kapsamında açıklayınız. Sizce neden önemli olabileceğini belirtiniz.**
Domain expert arasındaki ortak iletişimi sağlamakta, sağlanması gerektiğini ifade etmektedir. Domain expert'ler, alanlarına dair her ne kadar derin ve yeterli bilgiye sahip olsalar da yazılım geliştirme hakkında hiçbir şey bilmiyor olabilirler. Aynı şey yukarıda gördüğümüz gibi bir yazılım geliştirici için de geçerli olabilir ve çalışılacak alana dair herhangi bir bilgi söz konusu olmayabilir. İşte böyle bir durumda DDD yazılım geliştiricisi ile domain expert'ler arasında her iki tarafında rahat anlaşabilmesi için ortak dil bulunmasını önermektedir

9) **Core Domain, Supporting Domain, Generic Domain kavramlarını DDD kapsamında açıklayınız.**
Core Domain: En önemli alt domaindir. İş için temelleri barındırır. Bu domain olmadan system ayakta kalamaz.
Supporting Domain: Core domainden daha az önemli olan domaindir. Bu domain olmadan sistem ayakta kalabilir.
Generic domain: Sistemin işini kolaylaştıran ancak işin özü olmayan domaindir. Genelde dışardan temin edilir.

10) **Anemic Domain Model ve Rich Domain Model kavramlarını kıyaslayarak açıklayınız**
Anemic Domain Modelde, Domain modeller getter setter torbaları haline gelmesi, businees logic bulundurmaması ve business logic in serviste bu model üzerine konuşlandırılmamış olması gerekiyor.
Rich Domain Model oluşturularak servis katmanının katmanlar arasındaki bağlantıyı sağlamakla sorumlu tutulması ve ince olması gerektiği yönünde. Eric Evans'ın yazdığı domain driven design kitabında da belirttiği gibi servis katmanının amacı domaini gerekli işlemleri çözmeye yöneltmeli ve uygulamanın katmanları arasında bağlantıyı sağlamalı. Servis katmanı ağır business logiç içermemeli ve zayıf olmalıdır.