1) Spring dışında dependency injection için kullanabileceğimiz framework'ler / kütüphaneler nelerdir? (Herhangi bir programlama dili için olabilir.)
   Castle Windsor: .NET and Silverlight için kullanılan bir control containeri
   İnversify: Javascript depedency injection library
   Dagger 2: Android için kullanılabilecek bir library
   Guice: Google tarafından çıkarılan DI library

2) @SpringBootApplication anotasyonu hangi anotasyonları kapsamaktadır? Görevlerini kısaca açıklayınız
   SpringBootApplication: @Target, @Retention, @Documented, @Inherited, @SpringBootConfigration, @EnableAutoConfigration, @ComponentScan Anotasyonlarını kapsamaktadır.
   **Target**: Hangi anotasyon interface'inin uygulanabilir olduğunu belirtmek için kullanılır.
   **Retention**: Anotasyonun ne kadar süre kullanılacağını belirtiyor. Runtime boyunca kullan gibi.
   **SpringBootConfiguration**: Configurationları otomatik bulmak için kullanılıyor mesela testleri
   **EnableAutoConfiguration**: İhtiyacımız olacak beanleri tahmin etmeye ve configure etmeye çalışmaktadır.
   **ComponetScan**: Configureation classlarını kullanmak için otomatik directiveleri tarama işlemini yapar.
   **Documented**: JavaDoc dokümantasyonunu generate ediyor
   **Inherited**: Anotasyonu kullanan Classı başka bir class inherit ederse Anotasyonda otomatikmen inherit ediliyor

3) @Primary, @Qualifier anotasyonlarının kullanım amaçlarını açıklayınız
   Primary birden fazla bean içinde default olanı belirtmek için kullanılır, eğer class başına yazılırsa default class attribute'ini kullanır.
   Qualifier: Spring'e kaç tane bean olduğunu önemsemeden sadece belirtilen bean'i kullanmasını söyler.
   Yani iki anotasyonda multiple bean durumlarında hangisinin önceliğinin yüksek olduğunu belirtir.

4) Convention over configuration kavramını seçtiğiniz bir örnek üzerinden açıklayınız.
   Geliştiricinin esnekliğini kaybetmeden vermesi gereken karar sayısını azaltmaya yönelik tasarım kalıplarından birisidir. Configuration'u en aza indirmeyi hedefliyor. Uygulama configurationlarını harici dosyalara yazmak yerine kod içine yazmak diyebiliriz. Örnek vermek gerekirse bir plug-in veya tema dosyasını ilgili konumuna atarsanız uygulama görür ve işleme alır. Siz tekrar buraya dosya attım onu işleme alın diye configuration yapmanız gerekmez.

5) Aspect Oriented Programlama nedir ? Avantajları ve dezavantajları nelerdir ?
   Uygulama geliştirirken readibility, maintability, reuseablility gibi pek çok kavram kullanmaktayız ama proje büyüdükçe kod blokları uzadıkça bu işlem zorlaşıyor. Bu yüzden kodu parçalara ayırıp tekrar tekrar kullanılabilir yapmamız ASP'nin temel amacı. Validation, Log, DI gibi çokça kullanılan işlemleri soyutlanan modüllerle yapmamız kodumuzun clean olmasını sağlar. Bunu Springde veya javada anotasyonlarla yapabiliyoruz.

6) SOLID prensiplerini kısaca açıklayınız. Sizce her koşulda bu prensipler çerçevesinde mi kod yazılmalıdır? Neden?
   Geliştirilen yazılımın readibility, maintainbility, reuseability'sini arttıran prensipler bütünüdür. Robert C. Martin tarafından öne sürülmüştür. ''İyi kod'' yazmak için bu prensipleri uygulamaya özen göstermeliyiz.
   S - Single Responsibility
   Bir sınıfın sadece yapması gereken bir işi olması gerekir. Birçok özelliği olmaması gerekmektedir.
   O - Open Closed Principle
   Bir sınıf yada fonksiyon halihazırda olan özelliklerini korumalı ve değişikliğe izin vermemelidir. Davranışını değiştirmeden yeni özellikler kazanabilir olması gerekmektedir.
   L - Liskov Substitution Principle
   Alt sınıflar, Parent sınıfların özelliklerini kullanmalılardır.
   I - Interface Segregation Principle
   Sorumlulukların hepsini tek bir İnterface'e toplamak yerine özelleştirilmiş Interfaceler kullanmamız gerekir. Gereksiz metodların kullanımı engellenmelidir.
   D – Dependency Inversion Principle
   Sınıflar arası bağımlılık azaltılmalıdır. Üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır, ilişkiler abstraction veya interface kullanılarak sağlanmalıdır.

   OOP projelerin hepsinde bu kalıpları uygulamamız gerektiğini düşünüyorum, Havacılıktaki kurallarla alakalı bir söylem vardır ''Havacılık kuralları kanla yazılmıştır'' bu prensiplerde ilerde daha büyük sorunlarla karşılaşmamamız için uygulamamamız gereken prensiplerdir.

7) Swagger nedir, ne amaçla kullanılmaktadır?
   Swagger,Rest API geliştirmek için gerekli sözleşme standartlarını ve yardımcı toolları sunan bir araç, yazdığımız API'ların dökümantasyonu kısmında bize yardım eder. İnsanların kaynak koda erişmeden RestApilerin özelliklerini görmesine ve incelemesine olanak tandır.

8) Richardson Maturity Model'i seviyeleriyle birlikte açıklayınız.
   Level-0 Swamp of POX (Plain Old XML)
   Sıfır seviyesindeki servis sadece bir URI ve http metoduna sahiptir genellikle POST, bu seviye transfer protokolu olarak da adlandirilmaktadir.
   Level-1 Resources
   Birinci seviyede servisimiz birçok URI ama sadece bir HTTP metoduna sahiptir, genellikle POST, sistemdeki birçok kaynağı kullanıcıya sunabilmek amacındadır.
   Level-2 HTTP Verbs
   İkinci seviyede servisimiz HTTP metodlarına sahiptir ve CRUD işlemlerini yapar. HATEOAS kullanılmamaktadır.
   Level-3 Hypermedia Control
   Servisimizde URI'nin istek gönderip cevaba göre tekrar URİ'ye istek yapabildigimiz bir seviyedir. HATEOAS kullandığımız seviyedir.

9) URL, URI, URN kavramlarını bir örnek üzerinden açıklayınız.
   URI(Uniform Resource Identifier): Resource'a işaret eden, bir resource'u ada veya konuma göre tanımlar ve bu işaretleme için standart bir formata sahitir. URL ve URN, URI nin alt kümeleri olarak tanımlayabiliriz.
   Örnek: http://örnek.com/örnek.png

URL(Uniform Resource Locator): Resource'u tanımlayan ve ulaşmamız için gereken bilgiyi içeren bir URI dir. Her URL bir URI dir ancak her URI bir URL değildir.
URN(Uniform Resource Name): Resource'u unique bir isimle tanımlar fakat konumunu belirtmez.
Örnek: tel:+90-111-111-1111

10) Idempotency nedir ? Hangi HTTP metotları idempotent' tir ?
Idempotent birçok kez çağırıldığında alınan sonucun aynı olan metodlardır.
GET: metodu sunucuya id'si 1 olan memberları getir dediğimizde hep aynı sonucu getireceği için idempotentdir.
PUT: Sunucuya istek gönderdiğimizde update edecektir, bir kez daha yolladığımızda bir değişiklik olmayacaktır o yüzden idempotentdir
DELETE: Sunucuya istek yolladığımızda silme işlemini gerçekleştirecek, bir daha yolladığımızda ise herhangi bir işlem yapmayacaktır o yüzden idempotentdir.
POST: Sunucuya istekte bulunduğumuzda create işlemi yapacaktır birçok kez yaparsak birçok create işlemi yapacaktır ve sonuçlar farklı olacaktır. O yüzden idempotent değildir.

11) RFC (Request For Comment) neyi ifade etmektedir? HTTP hangi RFC dokümanında açıklanmıştır? Bu dokümanda HTTP hakkında ne tür bilgiler yer almaktadır?
RFC internet standartlarını ve protokolleri için teknik yayınlar olarak tanımlayabiliriz.
HTTP 1.0 RFC 1945 de yayınlandı. HTTP 1.1 protokolü ilk olarak 1997'de RFC2068'de yayınlandı. Bu belge 1999'da RFC2616 gelmesi ile iptal edildi ve aynı şekilde 2014'te. RFC 7230 ile değiştirildi.
RFC2616 toplam 20 ana başlık ve bir çok alt başlıklardan oluşmaktadır. Purpose, Terminology, Basic Rules, General Syntax gibi fundamentals konuları açıklıyor. Media types, header, body, length, request, response gibi ana önemli bilgilere yer verilmiş. Hata kodları ve anlamları açıklanmış.