Theoretical Homework 1

1 – IOC and DI means?

Inversion of Control (IoC) birbirine bağımlılığı az olan (loose coupled) nesneler oluşturmayı amaçlayan bir yazılım geliştirme prensibidir. Nesnelerin yaşam döngüsünden ve kontrol edilmesinden sorumludur.

Dependency Injection (DI) IoC prensiplerine uyarak birbirinden bağımsız nesneler oluşturmayı, herhangi bir interface üzerinden çalışıldığında yeni oluşturulan nesnelerin referanslar üzerinden oluşturularak loose coupling olarak kullanılıp, herhangi bir değişiklikte birbirinden etkilenmeyecek yapılar oluşturmayı amaçlar.

2 – Spring Bean Scopes?

Oluşturulan bean'lerin kapsamlarını tanımlar ve IoC container üzerinden yönetilir. Singleton, prototype, request, session ve globalSession gibi türleri vardır.

3 – What does @SpringBootApplication do?

Bir Spring uygulamasını önyüklemek ve başlatmak için kullanılır. ApplicationContext'i otomatik oluşturup yapılandırma sınıflarını tarar ve uygulamayı başlatır.

4 – Why Spring Boot over Spring?

Spring Boot teknolojisi genel olarak Spring kullanarak oluşturulan bir projede izlenen yolların derlenip toplanıp bir araya getirilerek kullanıma sunulması olarak değerlendirilebilir. Spring tarafındaki iş gücünün Spring Boot sayesinde azaltılmış olması, manuel olarak eklenen serverların veya Autoconfiguration gibi yapıların Boot ile beraber daha basitleştirilerek ve bir yapının içinde kullanıcıya sunulması Spring Boot'un en büyük artılarındandır.

5 – What is Singleton and where to use it?

Spring'teki genel scope'tur. Singleton sınıfından oluşturulan bir nesne bir kez üretilir ve her çağrıldığında aynı nesne gönderilir. Sabit bir nesne kullanmak istediğimizde Singleton kullanarak nesneyi bir kere üretiriz ne zaman ihtiyacımız olursa öncesinde ürettiğimiz nesneye ulaşarak her defasında nesne üretmemize gerek kalmamış olur.

6 – Explain @RestController annotation in Sprint boot?

@Controller ve @ResponseBody annotation'larının yerine kullanılan ve MVC 4.0'dan sonra kullanılmaya başlanılan annotation'dır. İsteklere cevap verecek method için sadece mapping işlemi yapmak yeterli olacaktır.

7- What is the primary difference between Spring and Spring Boot?

Spring Java EE kullanır, Spring Boot REST API'lerini kullanır.

Spring Boot gömülü serverlara sahiptir, Spring'de bunlar yoktur.

Dependencyler Spring projelerinde manuel olarak yapılması gerekirken Spring Boot bunu kendisi halleder.

Spring Boot çok daha az kod yazarak istenileni yapabilir, Spring daha karmaşıktır.

Web uygulamaları geliştirmek Spring Boot ile çok daha kolaydır.

8 – Why to use VCS?

Geliştirilen projelerin tüm dosyalarına ya da belli başlı dosyalarına ulaşmamızı, geçmişte yapılan versiyonlarına erişmemizi, yapılan değişiklikleri ve kim tarafından yapıldığını görmemizi sağlayan sistem olarak VCS kullanır. Böylelikle projelerin ilk andan son haline kadar, aşama aşama takibinin yapılması ve değişikliklerin yönetilmesi oldukça kolaylaşır.

9 – What are SOLID Principles? Give sample usages in Java?

SOLID nesne yönelimli programlamanın beş temel prensibidir.

S: Single Responsibility Principle (Tek Sorumluluk İlkesi)

- Her sınıfın ya da benzer yapının tek bir sorumluluğu olmalıdır.

O: Open-Closed Principle (Açık-Kapalı İlkesi)

- Sınıflar genişletilmeye açık ancak değiştirilmeye kapalı olmalıdır.

L: Liskov Substitution Principle (Liskov Değişim İlkesi)

- Bir sınıftan türeyen tüm sınıflar (alt sınıflar), üst sınıfın tüm özelliklerini gösterebilmeli ve genişletebilmelidir.

I: Interface Segregation Principle (Arayüz Ayrıştırma İlkesi)

- Tek bir interface'e birçok özellik eklemek yerine her özellik için ayrı interface oluşturmak ve bütünü parçalara ayırmak gerekir.

D: Dependency Inversion Principle (Bağımlılığı Tersine Çevirme İlkesi)

- Sınıflar arası bağımlılıklar olabildiğince az olmalıdır. Özellikle üst sınıflar alt sınıflara kesinlikle bağımlı olmamalıdır.

Tek Sorumluluk İlkesi

```
public class Vehicle {
   public void details() {}
   public double price() {}
   public void addNewVehicle() {}
}
```

Burada üç farklı görevi olan Vehicle sınıfının bu şekilde kullanımı uygun değildir.

Tek sorumluluk ilkesi hedefine ulaşmak için, yalnızca tek bir işlevi gerçekleştiren ayrı bir sınıf uygulamalıyız.

```
public class VehicleDetails {
   public void details() {}
}

public class CalculateVehiclePrice {
   public double price() {}
}

public class AddVehicle {
   public void addNewVehicle() {}
}
```

Açık-Kapalı Prensibi

Bu prensibi bir bildirim hizmeti örneği ile anlayalım.

```
public class NotificationService{
   public void sendNotification(String medium) {
      if (medium.equals("email")) {}
   }
}
```

Burada e-posta dışında yeni bir ortam tanıtmak istiyorsanız, diyelim ki bir cep telefonu numarasına bildirim gönderin, o zaman NotificationService sınıfında kaynak kodunu değiştirmeniz gerekiyor.

Bunu aşmak için kodunuzu, herkesin özelliğinizi genişleterek yeniden kullanabileceği ve herhangi bir özelleştirmeye ihtiyaç duyarsa sınıfı genişletip üzerine özelliklerini ekleyebileceği şekilde tasarlamanız gerekir.

Aşağıdaki gibi yeni bir arayüz oluşturabilirsiniz:

```
public interface NotificationService {
   public void sendNotification(String medium);
}
```

Eposta bildirimi:

```
public class EmailNotification implements NotificationService {
   public void sendNotification(String medium){
      // write Logic using for sending email
   }
}
```

Mobil Bildirim:

```
public class MobileNotification implements NotificationService {
   public void sendNotification(String medium){
      // write Logic using for sending notification via mobile
   }
}
```

Liskov İkame Prensibi

Rectangle temel sınıfından türetilen bir kare sınıf örneğini ele alalım:

```
public class Rectangle {
   private double height;
   private double width;
   public void setHeight(double h) { height = h; }
   public void setWidht(double w) { width = w; }
}
public class Square extends Rectangle {
   public void setHeight(double h) {
      super.setHeight(h);
      super.setWidth(h);
   }
   public void setWidth(double w) {
      super.setHeight(w);
      super.setWidth(w);
   }
}
```

Rectangle Sınıfta genişlik ve yükseklik ayarı son derece mantıklı görünüyor . Ancak square sınıfında SetWidth() ve SetHeight() anlamlı değildir çünkü birini ayarlamak diğerini ona uyacak şekilde değiştirir.

Bu durumda, Rectangle temel sınıfını türetilmiş Square sınıfıyla değiştiremeyeceğiniz için Square, Liskov ikame testinde başarısız olur. Square sınıfının ekstra kısıtlamaları vardır, yani yükseklik ve genişlik aynı olmalıdır. Bu nedenle, Rectangle sınıfının Square sınıfıyla değiştirilmesi beklenmeyen davranışlara neden olabilir.

Arayüz Ayrıştırma İlkesi

Bunu bir araç arayüzü örneği ile anlayalım:

```
public interface Vehicle {
   public void drive();
   public void stop();
   public void refuel();
   public void openDoors();
}
```

Bike, Şimdi bu Araç arayüzünü kullanarak bir sınıf oluşturduğumuzu varsayalım.

```
public class Bike implements Vehicle {
   public void drive() {}
   public void stop() {}
   public void refuel() {}

   // Can not be implemented
   public void openDoors() {}
}
```

Bisikletin kapıları olmadığı için son işlevi uygulayamıyoruz.

Bunu düzeltmek için, hiçbir sınıfın ihtiyaç duymadığı herhangi bir arabirimi veya yöntemi uygulamak zorunda kalmaması için arabirimleri küçük çoklu arabirimlere ayırmanız önerilir.

```
public interface Vehicle {
   public void drive();
   public void stop();
   public void refuel();
}
public interface Doors{
   public void openDoors();
}
```

İki sınıf oluşturma - Car ve Bike

```
public class Bike implements Vehicle {
   public void drive() {}
   public void stop() {}
   public void refuel() {}
}
public class Car implements Vehicle, Door {
   public void drive() {}
   public void stop() {}
   public void refuel() {}
   public void openDoors() {}
}
```

Bağımlılık Tersine Çevirme İlkesi

Müşterilerin en sevdikleri kitapları belirli bir rafa koymalarını sağlayan bir kitapçı olduğunu varsayalım.

Bu işlevi uygulamak için bir Book sınıf ve bir Shelf sınıf oluşturuyoruz. Kitap sınıfı, kullanıcıların raflarda depoladıkları her kitabın incelemelerini görmelerine olanak tanır. Raf sınıfı, raflarına bir kitap eklemelerine izin verecektir. Örneğin,

```
public class Book {
   void seeReviews() {}
}


public class Shelf {
    Book book;
    void addBook(Book book) {}
}
```

Her şey yolunda görünüyor, ancak high-level Shelf sınıf low-level'e bağlı Book olduğundan, yukarıdaki kod Dependency Inversion Principle'ı ihlal ediyor. Bu, mağaza bizden müşterilerin kendi yorumlarını da raflara eklemelerini sağlamamızı istediğinde açıkça ortaya çıkıyor. Talebi karşılamak için yeni bir UserReview sınıf oluşturuyoruz:

```
public class UserReview{
    void seeReviews() {}
}
```

Şimdi, Kullanıcı İncelemelerini de kabul edebilmesi için Shelf sınıfını değiştirmeliyiz. Ancak bu, Açık/Kapalı İlkesini de açıkça bozacaktır.

Çözüm, alt düzey sınıflar (Kitap ve UserReview) için bir soyutlama katmanı oluşturmaktır. Bunu Ürün arabirimini tanıtarak yapacağız, her iki sınıf da onu uygulayacak. Örneğin, aşağıdaki kod kavramı gösterir.

```
public interface Product {
   void seeReviews();
}

public class Book implements Product {
   public void seeReviews() {}
}

public class UserReview implements Product {
   public void seeReviews() {}
}
```

Artık Raf, uygulamaları yerine Ürün arayüzüne başvurabilir (Kitap ve Kullanıcı İncelemesi). Yeniden düzenlenmiş kod ayrıca, müşterilerin raflarına da koyabilecekleri yeni ürün türlerini (örneğin, Dergi) daha sonra tanıtmamıza da olanak tanır.

```
public class Shelf {
   Product product;
   void addProduct(Product product) {}
   void customizeShelf() {}
}
```

10 - What is RAD model?

Hızlı uygulama geliştirme modeli olarak Türkçeleştirilebilir bu modelde öncelikle gereksinimler belirlenir. Ardından analiz yapılıp hızlıca dizayna geçilir. Dizayn kısmı da tamamlandıktan sonra implementasyonlar yapılır ve proje test edilir. Testlerin sonuçlarına göre değişiklikler yapılacaksa adımlar tekrarlanır.

11 - What is Spring Boot starter? How is it useful?

Spring Boot starterlar projelere kullanmak istediğimiz teknolojileri eklememize yarayan araçlardır. Starterlar sayesinde tek tek arama yapıp ekleme derdinden kurtulmuş oluruz. Örnek olarak Spring projemizde JPA kullanmak istiyorsak “spring-boot-starter-data-jpa” bağımlılığını projemize ekleyerek bunu gerçekleştirebiliriz.

12 – What are the Spring Boot Annotations?

Spring Boot Annotations, bir program hakkında veri sağlayan bir meta veri biçimidir. Geliştirdiğimiz uygulamanın bir parçası değildir ve açıklama yaptıkları kodun çalışması üzerinde doğrudan bir etkisi yoktur. Derlenen programın faaliyetlerini etkilemez.

Başlıca kullanılanlara örnek verecek olursak:

@Bean

@Service

@Repository

@Configuration

@Controller

@RequestMapping

@Autowired

@Component

13 – What is Spring Boot dependency management?

Spring Boot'un sahip olduğu ve kendi kendini yönetebilen bağımlılık sistemidir. Spring Boot'un her sürümü otomatik olarak kendi bağımlılık listesini oluşturur ve bunu yönetir. Versiyon belirtme ihtiyacı yoktur.

14 - What is Spring Boot Actuator?

Çalıştırdığımız Spring Boot uygulaması hakkında endpointler yardımıyla bilgi (projenin bilgileri, çalışıp çalışmadığı, spring tarafından yönetilen beanler, trafik durumu vb) almamızı sağlar. Pom.xml dosyasına eklenecek bir bağımlılık ile bu bilgilere ulaşabiliriz. Standart olarak iki adet endpointe erişim sağlayabiliriz ancak yapılacak düzenlenmelerle bu arttırılabilir.