2. HAFTA JAVA BOOTCAMP ÖDEVİ

1) Creational Design Pattern'lar incelenmelidir. Örneklerle anlatınız.

Singleton Pattern: Bir sınıfın sadece bir örneğinin olmasını sağlar. Singleton nesneye istediğimiz yerde ulaşabilmenizi sağlar.

```
public class SingletonDesignPattern {

   private static SingletonDesignPattern singletonDesignPattern = new SingletonDesignPattern();

   private SingletonDesignPattern() {

   }

   public static SingletonDesignPattern getSingletonDesignPattern() {

      return singletonDesignPattern;

   }

}

public class Main {

   public static void main(String[] args) throws Exception {

     SingletonDesignPattern singletonDesignPattern =
SingletonDesignPattern.getSingletonDesignPattern();

   }

}
```

Factory Pattern: Nesne oluşturmak için bir arayüz tasarlanmasını gerektirir ve alt sınıfların nesne üretmesini sağlar.

```
public interface Computer {

 void showModel();

}

public class Asus implements Computer {

   @Override

   public void showModel() {

      System.out.println("Asus ROG Strix G513IC-HN037");

   }

}

public class Acer implements Computer {

  @Override
```

```
    public void showModel() {
        System.out.println("Acer Aspire 3 A315-57");
    }
}
public class ComputerFactory {
    public Computer getComputer(String computerModel){
        if(computerModel == null) {
            return null;
        }
        if(computerModel.equalsIgnoreCase("ACER")) {
            return new Acer();
        }
        else if(computerModel.equalsIgnoreCase("ASUS")) {
            return new Asus();
        }
        return null;
    }
}
```

Abstract Factory Pattern: Aynı anda birden fazla nesne ile işlem yapmak istediğimiz durumlarda kullanılır.

Builder Pattern: Oluşturduğumuz nesnenin içerisindeki değişkenlere bağlı olarak oluşan constructorda kullanmadığımız, gereksiz parametreler olabilir. Bu design pattern sadece istediğmimiz bir parametreyi almasını sağlayacağımız bir constructor yaratmamıza imkan sunar.

Prototype Pattern: Nesnemizi birden fazla oluşturmamız gereken durumlarda new olarak oluşturmak yerine nesnemizin kopyasını oluşturmamızı sağlar.

2) Derste yapılan emlakcepte uygulamasını Singleton ve Factory pattern'larını uygulayın.

Oluşturulan objeler bu pattern'ler ile oluşturulmalıdır.(Spring içerisinde BeanFactory gibi davranmasını bekliyoruz)

```
private static RealtyService realtyService = new RealtyService();

private RealtyService(){

}

private static RealtyService getInstance(){

        return  new RealtyService();

        }
```

4) Java dünyasındaki framework'ler ve çözdükleri problemler nedir? Kod Örneklendirini de içermelidir.

Frameworkler, yazılımı geliştirmek, desteklemek ve daha az kodla başarılı bir program ortaya çıkarmamızı sağlayan araçlardır.

Java dünyasındaki frameworkler ise; **SPRING,JSF (Java Server Faces),MAVEN,HIBERNATE,JSP**'dir.

Spring: Java dilinde en çok kullanılan frameworklerden biridir. MVC katmanlaarını kontrol eder, karmaşık bir sistemi dahada basitleştirmemizi sağlar. Kurumsal ortamlarda daha çok tercih edilir.

```
<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
   </dependency>
</dependencies>
```

```
@RestController
@RequestMapping("/api/languagetechnology")
```

```
public class LanguageTechnologyControllers {
    private LanguageTechnologyService languageTechnologyService;

    @Autowired
    public LanguageTechnologyControllers(LanguageTechnologyService languageTechnologyService){
        this.languageTechnologyService = languageTechnologyService;

    }
    @GetMapping("/getall")
    public List<GetAllLanguageTechnologysResponse> getAll() {
        return languageTechnologyService.getAll();
    }

    @PostMapping("/add")
    public void add(@RequestBody() CreateLanguageTechnologyRequest
createLanguageTechnologyRequest){ //gelen istekleri java sınıfıyla eşlemek için requestbody kullanılır
        this.languageTechnologyService.add(createLanguageTechnologyRequest);
    }

    @DeleteMapping("/{id}")
    public void delete(int id) {
        languageTechnologyService.delete(id);
    }

    @PutMapping("/update")
    public void update(@RequestBody() UpdateLanguageTechnologyRequest
updateLanguageTechnologyRequest, int id) throws Exception {
        this.languageTechnologyService.update(updateLanguageTechnologyRequest, id);
    }

    @GetMapping("/{id}")
    public Optional<LanguageTechnology> getById(@PathVariable() int id){
```

```
    return languageTechnologyService.getById(id);
  }
}
```

JSF: Web uygulamalarını geliştirmek için kullanılır. Kullanıcı arayüzü bulunur. Kullanıcı arayüzünün oluşturulmasını sağlar.

```
<html>
<h:head>
  <title>JSF</title>
</h:head>
<h:body>
  <h:form>
    <h:inputText value="#user.name}" />
    <h:inputText value="#{user.age}" />
    <h:inputText value="#{user.mail}" />
   <h:commandButton value="Gönder" action="#{user.save}" />
  </h:form>
</h:body>
</html>
```

Maven: Projenin bağımlılıklarını tanımlayan bir pom.xml kullanır. Bağımlılıkları yönetmemizi ve derleme işlemini kolaylaştırır. IDE bağımlılığı yoktur.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.7.5</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
```

```
	<groupId>kodlama.io</groupId>
	<artifactId>kodlamaioDevs2</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>kodlamaioDevs2</name>
	<description>Demo project for Spring Boot</description>
```

Hibernate: Veritabanı işlemlerinde kullanılır.

```
@Table(name = "technologys")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class LanguageTechnology {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "technology_id")
   private int id;
   @Column(name = "technology_name")
   private String technologyName;
   @ManyToOne
  @JoinColumn(name = "language_id")
   public ProgrammingLanguage programmingLanguage;
}
```

JSP: HTML kodları içerisine java komutlarını kolay bir şekilde yerleştirmek için kullanılır. Java kodları sayfanın dinamik olarak oluşturulması ve güncellenmesi için kullanılır.

5) Spring frameworkünün kullandığı design patternlar neler?

Spring Framework içerisinde kullanılan design patternlar şunlardır:

Singleton: Bir sınıfın sadece bir örneğinin oluşturulmasını ve bu örneğe her yerden erişilebilmesini sağlar.

Factory Method: Nesne yaratmak için arayüz tasarımı oluşturmamızı sağlar.

Template Method: Sınıflar arasındaki kod tekrarını azaltır ve kodun daha okunabilir olmasını sağlar.

Prototype: Bir objeyi sınıflarına bağımlı olmadan kopyalamamızı sağlar.

Proxy: Başka bir nesneye erişimi kontrol etmesine izin verir.

6) SOA - Web Service - Restful Service - HTTP methods kavramlarını örneklerle açıklayınız.

SOA: Servis odaklı mimaridir. SOA'nın amacı uygulamaları birlikte çalışabilir hale getirmek ve standart protokoller üzerinden iletişim kurmalarını sağlamaktır. Esneklik, kolay bakım, yeniden kullanılabilirlik, uyumluluk sağlar.

Web Service: HTTP protokolü üzerinden diğer sistemlere hizmet verir. Farklı platformlar arasında veri alışverişi yapan bir yapıdır. Veri trafiğini platformdan bağımsız gerçekleştirilir. İki çeşit web service vardır. REST VE SOAP.

Restful Service: Genellikle HTTP protokolü üzerinden çalışırlar. GET,POST,PUT,DELETE metotlarını kullanır.

HTTP methods: Kullanıcı sunucuya bir istek (request) gönderir ve sunucu bir cevap (response) döner.

GET: Verileri görüntülemek için kullanılır.

POST: Veri eklemek için kullanılır.

PUT: Veri güncellemek için kullanılır.

DELETE: Veriyi silmek için kullanılır.

PATCH: Belirli bir değişikliği yapmak için kullanılır.