K147. Kodluyoruz Java Bootcamp
Hafta 2 - Ödev

2) Creational Design Pattern'ları objelerin oluşturulmaları şekli ile ilgilidir, bunlar :
(Patternlara ait örnek kodlar github'a yuklenmiştir.)

**Singleton** → Bir sınıftan sadece bir tane nesnenin bulunması gereken durumlara bir çözümdür. Aşagıda görüldügü Singleton class'indan nesne oluşturmak için getInstance metodu kullanılmalıdır. Static oldugu için bir kez olusturulur ve Constructor private durumdadır , new keyword'u ile dışarıdan instance alınması engellenmiştir.

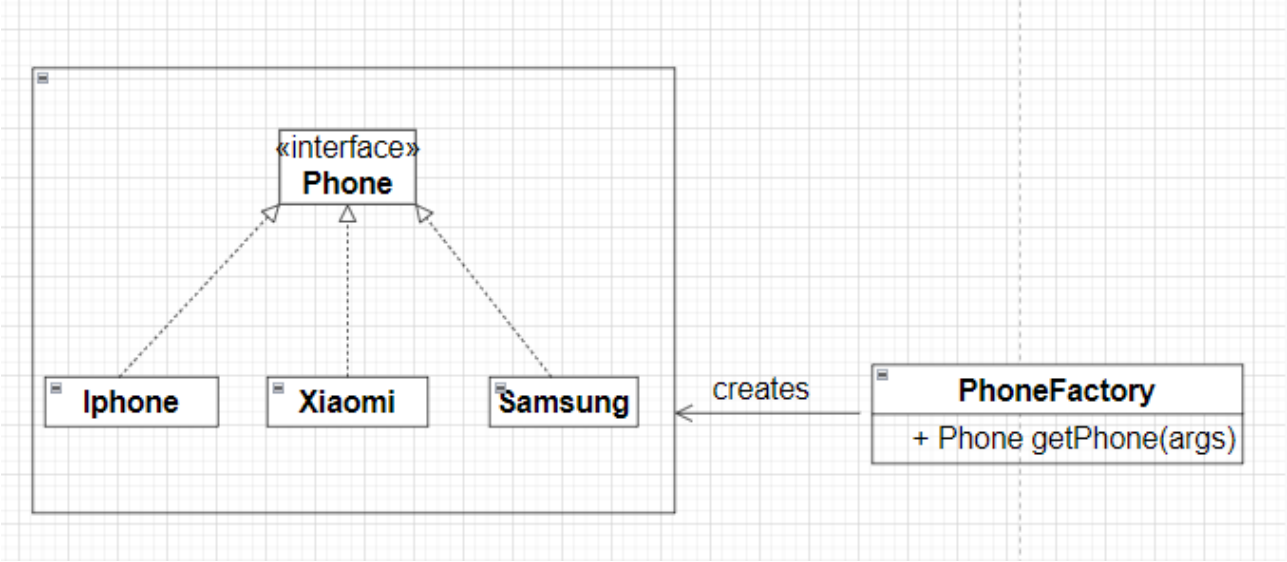
```java
package com.works.creationalpatterns.singleton;

public class Singleton {
    /// Objeye ait tek INSTANCE property'si olacak.
    private static Singleton INSTANCE = new Singleton();

    // constructor ile instance olusturulmasını onlenir
    private Singleton() {}

    // Class uzerinde erişim izni var.
    public static Singleton getInstance() {
        return INSTANCE;
    }
}
```

**Prototype** → Elinde bir nesne var ve bu nesnenin birebir kopyasını bir çok kez daha olusturmak zorunda isek, sıfırdan olusturmak yerine elinde var olan nesnenin clonelarını al. Yani elimizdeki nesne prototip oluyor, diğer nesneler de bu prototip üzerinden aynı özellklere sahip olarak üretiliyor.
Aşagıda Customer nesnesi Prototype interface'ini implemente etmetedir. Ona ait olan getClone() metodunu implemente etmektedir yaptıgı iş oluşturulan Customer nesnesinin aynı özelliklere sahip bir kopyasını oluşturmaktadır.

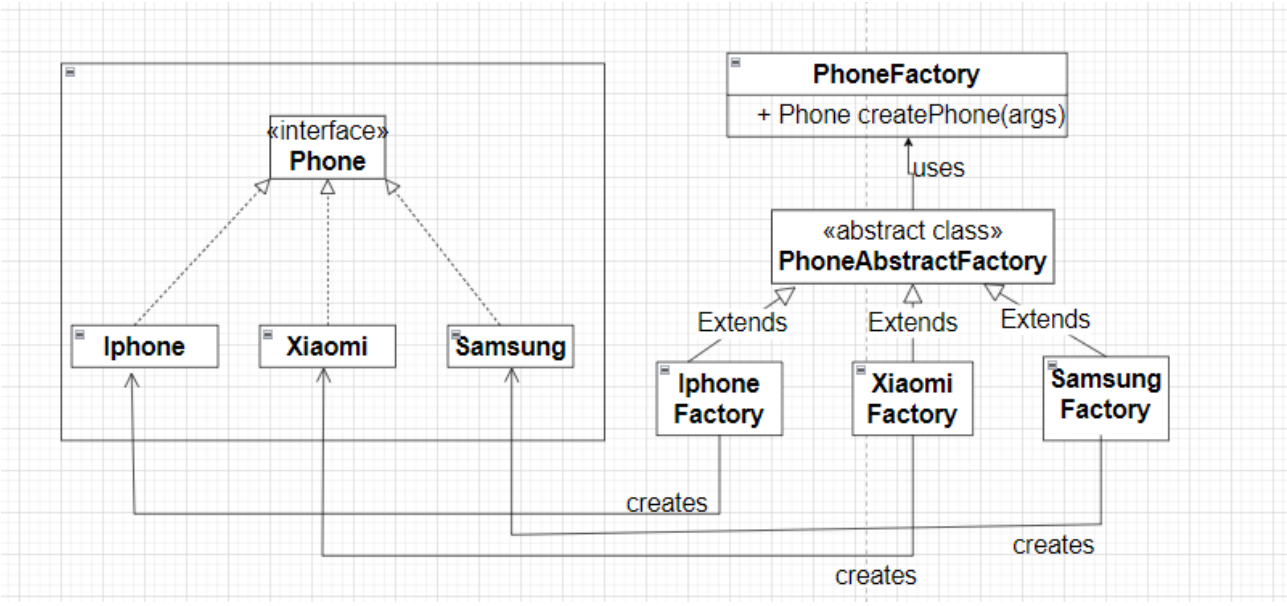
Customer.java

```java
package com.works.creationalpatterns.prototype;

public class Customer implements Prototype{

    private int id;
    private String username;
    private int age;
    private String address;

    public Customer() {}

    public Customer(int id, String username, int age, String address) {
        this.id = id;
        this.username = username;
        this.age = age;
        this.address = address;
    }

    public void showInfo() {
        System.out.format("%-4d  %15s %5d %10s\n", id,username,age,address);
    }


    @Override
    public Prototype getClone() {
        return new Customer(id,username,age,address);
    }

}
```

**Factory Method** →Üst sınıfta nesneler oluşturmak için bir arabirim sağlayan, ancak alt sınıfların oluşturulacak bu nesne türünü değiştirmesine izin veren bir creational pattern türüdür.
Factory Method deseni doğrudan nesne oluşturma çağrılarını (`new` operatörü kullanarak), özel bir fabrika metodu çağrısına dönüştürmeyi öneriyor, getPhone metoduna istenilen tip girilerek istenilen nesene olusturulabilir




**Abstact Factory** → Birbiriyle alakalı veya bağımlı nesnelerin somut sınıflarını belirtmeden, yaratılması için gereken bir arayüz sağlar. Nesnenin üretimini tek bir arayüz tarafından değil her sınıf için farklı bir arayüz tanımlayarak sağlamaktadır.
Asagıda görüldugu gibi Factory metodundan fark olarak burada her sınıfa ait factory sınıfı bulunmaktadır.PhoneFactory'ye hangi sınıfa ait Factory medotu verilirse onun uretmesi gerektigi sınıfın instance'ini üretir.



**Builder** → Karmaşık nesneleri adım adım oluşturmanıza olanak tanır. Kalıp, aynı construction kodunu kullanarak bir nesnenin farklı türlerini ve temsillerini oluşturmanıza olanak tanır.

Sınıfımızda bulunan field sayısı ne kadar fazla olursa bundan dolayı yapılacak constructora sayısı artar. Çünkü nesneyi oluştururken hangi field başta atama yapılacak ya da yapılmayacak bilemeyebiliriz. İşte bu uzayıp giden parametre sayısından, karmaşık constructorlardan kurtarmak için builder pattern kullanılır.Asagıdaki örnekte 5 tane field'a sahip olan Employee sınıfının builder sahip oldugu ve builder sınıfına olmadan ne kadar fazla karmaşaya sahip oldugu görülmektedir.

Employee.java

```java
package com.works.creationalpatterns.builder;

public class Employee {

    private String firstName; // required
    private String lastName; // required

    private int age; // optional
    private String phone; // optional
    private String address; // optional

    public Employee(EmployeeBuilder builder) {
        this.firstName = builder.firstName;
        this.lastName = builder.lastName;
        this.age = builder.age;
        this.phone = builder.phone;
        this.address = builder.address;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
```

EmployeNotHaveBuilder.java

```java
    public EmployeNotHaveBuilder() {}
    public EmployeNotHaveBuilder(String firstName, String lastName, int age, String phone, String address) {
    public EmployeNotHaveBuilder(String lastName, int age, String phone, String address) {}
    public EmployeNotHaveBuilder(int age, String phone, String address) {}
    public EmployeNotHaveBuilder(String firstName, String lastName, int age) {}
    public EmployeNotHaveBuilder(String firstName, String lastName,String phone, String address) {}
    public EmployeNotHaveBuilder(String firstName, String lastName, int age, String phone) {}
    public EmployeNotHaveBuilder(int age,String firstName, String lastName, String phone) {}
    public EmployeNotHaveBuilder(String firstName, String lastName, String phone) {}
    public EmployeNotHaveBuilder(String firstName,int age) {}
    public EmployeNotHaveBuilder(String firstName, String lastName) {}
```

4) Yaptıgım uygulamada maven-hibernate-spring kullanılmıştır. Mysql veritabınında yer alan world verisi kullanılmaktadır. Kaynak kodlar github'a yüklenmiştir

**HİBERNATE** → ORM kütüphanesidir. Nesne Yönelimli modellere göre veritabanı ile olan ilişkiyi sağlayarak, veritabanı üzerinde yapılan işlemleri kolaylaştırmaktadır.Java sınıflarının veritabanı dönüşümünü yapmaktadır.

**MAVEN** → Proje geliştirirken proje içerisinde bir standart oluşturmamızı, geliştirme sürecini basitleştirmemizi, dokümantasyonumuzu etkili bir şekilde oluşturmamızı, projemizdeki kütüphane bağımlılığını ve IDE bağımlılığını ortadan kaldırmamızı sağlayan bir araçtır.

**SPRİNG** → Java tabanlı enterprise uygulamalar için kapsamlı bir programlama ve konfigürasyon altyapı desteği sunar.Bu sayede yazılımcı business logic'e (verinin yorumlandığı ve iş kurallarının uygulandığı) katmana odaklanabilir. MVC mimarisi,Aspect Oriented Programlama imkanı sağlaması, Restful web servisleri sağlaması, JDBC, JPA desteği gibi birçok özellik sağlamaktadır.

**JSF** → Java Server Faces, web sayfalarında Java dilini kullanarak dinamik web sayfaları oluşturmamızı sağlayan bir Java teknolojisidir.

5) Singleton, Prototype, Factory, Template Design pattern, MVC Pattern, Proxy Pattern, Observer Pattern, Dependecy Injection(DI), Front Controller

Spring de bir bean olusturunca bunun kapsamını(Scobe) belirtebiliyoruz burada singleton veya prototype seceneklerini secebiliriz istenilen secenege göre o pattern kullanılır.

Factory pattern ise BeanFactory ve Application context kullanarak beanleri yüklemek için kullanılır
Örnek :

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(BeanConfig.class);
```

Templete degisn pattern ise tekrarlayan kodla başa çıkmak için kullanılır. Örnegin JdbcTemplate sınıfında kullanılır.Baglantı kurma , sorguyu çalıştırma, baglantıyı kapatmak gibi işlemler.

MVC Pattern, Spring MVC tarafından kullanılır. Model, view ve controller katmanlarını birbirinden ayırır.

Proxy pattern bir nesnenin başka bir nesneye erişimi konteol etmesine izin verir. @Transactional anotasyonu kısmında proxy pattern kullanılır..

Observer tasarım deseni, bir nesne kümesi arasındaki one-to-many ilişkiyi tanımlar. Bir nesnenin durumu değiştiğinde, bütün bağımlılarına bildirilir. ApplicationContext'in event mekanizmasında kullanılır. *@EventListener* anotasyonu ile kullanılır. *ContextStartedEvent, ContextRefreshedEvent,* ContextStoppedEvent standard eventlardır.

DI ise bagımlılıkların enjekte edilmesinde kullanılır, @Autowired keyword'u ile bagılılıklar contructor,setter veya properties yolu ile enjekte edilir.

Front Controller Design Pattern'in amacı, client'lardan gelecek olan istekleri merkezi bir yerde karşılayıp sonrasında ilgili kod parçacıklarına yönlendirmektir.
Spring MVC'de DispatcherServlet Front-Controller olarak kullanılır. DispatcherServlet tüm HTTP isteklerini cevaplayan, yöneten bir servlet mekanizmasıdır.

6) **Web Service** → Verilerinizi web sayfanız dışında tüm cihazlara göndermek istediğinizde devreye Web Service kavramı girer. Platform bağımsız tüm cihazlara veri aktarımı web service aracılıgıyla gerçekleştirilir.
Örnek olarak Facebook üzerinden mesajlaşırken mesajların hem web sayfasına hemde facebook messenger uygulamasına gelmesini web service saglar.

**Restful Web Service** → REST mimarisi temel alınarak geliştirilmiş oldukça hafif, genişletilebilir ve basit servislerdir. Restful servislerin amacı client-server arasındaki veri akışını platform bağımsız olarak gerçekleştirebilmek ve veri akışını en az yükle sağlayabilmektir.
Restful servisleri response tipi olarak JSON, HTML, XML gibi bir çok formatta çalışabilirler.
RESTful servisleri üzerinden CRUD (Create, Read, Update, Delete) işlemlerinin gerçekleştirilebilmesi için HTTP metotları kullanılır.
Örnek olarak *https://restcountries.com/v3.1/all* restful api'si ile ülkelere ait bilgilere ulaşabiliriz.

**Http methods** → Genel olarak kullanılan metodlar :

GET : Bu metod sunucudan veri almak için kullanılır. Herhangi bir siteye girdigimizde veya api'den üzerinden bilgi aldıgımızda bu metod çagrılır.

POST : Belirli bir kaynağa veri göndermek için kullanılır.
Bir siteye kayıt oldugumuzda veya giriş yapmak istegimizde post metodu kullanılır.Get metodu yerine Post metodu tercih edilir bilgilerimiz request body içine yerleştirilir.

PUT : Belirli bir kaynaktaki verinin değiştirilmesi için kullanılan metodtur.
Örnek olarak Veri tabanından bir bilginin güncellenmesi durumda kullanılabilir.

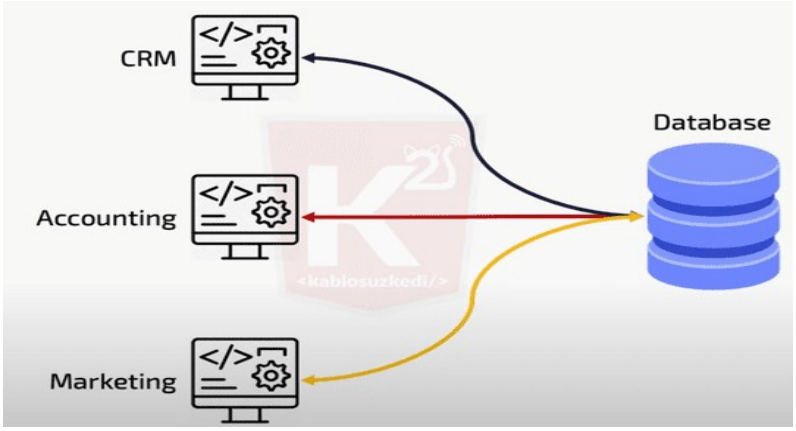
DELETE : Belirli bir kaynaktaki verilerin silinmesi için kullanılan metodtur.
Örnek olarak Veri tabanından bir bilginin silinmesi durumda kullanılabilir.

Patch: Bu metot da sunucudaki bir kaynağı değiştirmek için kullanılır. Put ile arasındaki fark ise Put sunucudaki kaynağı yeni bir kaynak ile değiştirmek için kullanılır iken, Patch bu kaynağında bir kısmını değiştirmeye yarar.

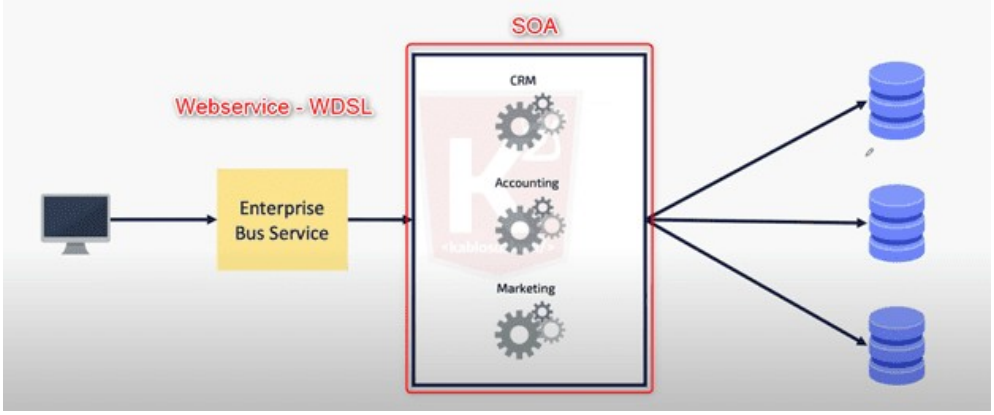
**Service Oriented Architecture (SOA)** → Hizmet odaklı mimari , birden fazla servisin ayrı ayrı tasarlandığı ve servislerin hem kendi aralarında hem de farklı DB ler ile haberleşebildiği mimaridir.

SOA servislerin ayrı ayrı tasarlanıp tek bir yapı oluşturulmasına imkan sağlar .
Yapılar birbirinden bağımsız olarak çalıştırılabilir bu yönteme loose coupling denmektedir.
Kendi içerisinde birden çok bileşeni vardır.

SOA'da her hizmet ayrı bir servis olarak görev yapmaktadır, servisler ayrı ayrı veya aynı database gidebilir.



Birden fazla servis ve birden fazla DB için  diagram aşağıdaki gibi olacaktır




Enterprise Bus Service , Client'ten gelen isteği alır ve hangi servis için istek gönderildiyse client'i o servis ile buluşturur. Servislerin oluşturduğu kümeye ise SOA denir. SOA kitlesinin dış dünya ile haberleşmesi için Webservice ve WDSL yöntemleri kullanılmaktadır.

Örnek vermek gerekirse derste oluşturdugumuz emlakcepte-servisi buna örnektir. Emlakcepte-servis ve emlakcepte-banner-servis ayrı ayrı oluşturulmus ve tek bir yapıyı temsil etmektedir.