2. Creational Design Pattern'lar incelenmelidir. Örneklerle anlatınız.

Creational design patern oluşturucu yada yaratımsal olarak türkçeye çevrilebilir. Nesne oluştururken sıklıkla bazı problemler ile karşılaşıyoruz ve bu problemlerin genel çözümleri bu kategoride bulunuyor. Öncelikle Creational design paternlerin neler olduğuna bakacak olursak.

-Singleton Design Pattern

-Factory Design Pattern

-Abstract Factory Design Pattern

-Builder Design Pattern

-Prototype Design Pattern

Bu patternleri sırasıyla incelemeye başlayalım.

-Singleton Design Pattern

Kelime anlamına bakacak olursak tekil anlamına gelmektedir. Sınıfımızdan sadece 1 nesne oluşsun istiyorsak bu design patterni kullanabiliriz. Kısacası newleme işlemi sadece 1 kez yapılmalıdır.

Singleton Patterni örnekleyecek olursak;

```
public class Singleton {
   private static Singleton singleton;   //
   private Singleton(){
      System.out.println("Ben olustum");
   }

   public static Singleton getSingleton() {
      if (singleton==null){
         singleton=new Singleton();
      }
      return singleton;
   }
}
```

Öncelikle nesnemizin 1 kere oluşmasını istiyorsak aklımıza ilk olarak Constructor'ımızın private olması gerekmektedir. Fakat böyle olunca bir

sorunla karşılaşırız nesnemizi oluşturmak istediğimizde hata oluşur nesnemiz hiç oluşamaz. Nesnemizi kendi classında oluşturmamız gerekmektedir bunu classın içinde bir fonksiyonda gerçekleştirebiliriz fonksiyonumuz classda private static olarak tanımladığımız nesne yi döndürebilir ve bu fonksiyona dışarıdan direkt class ismi ile erişebilmek için static yapmalıyız. Böylelikle nesnemiz artık 1 kere oluşmaktadır. Fakat ortaya bir sorun daha çıkmıştır. Nesnemiz biz istesek de istemesekte herzaman 1 kere oluşuyor , bizim isteğimiz ise sadece biz istediğimizde 1 kez oluşması, Bunun için fonksiyonun içinde null kontrolü yapıp nesnemizi dışarıda değilde fonksiyonun içerisinde oluşturmalıyız böylelikle biz fonksiyonu çağırdığımızda nesne oluşacaktır.

-Factory Design Pattern

Örnek üzerinden anlatacak olursak. Kullandığımız telefonu düşünelim telefonu bayiler değil fabrikalar üretir bizde nesneyi oluşturma işlemini bayilere değil fabrikalara vereceğiz. Kod üzerinden örneklemeye çalışalım;

Öncelikle bir telefon interfacesi oluşturuyoruz.

```
public interface Telefon {
   String getModel();
   String getBatarya();
   int getEn();
   int getBoy();

}
```

Ardından telefon sınıfımızı Telefon interfacesinden implement alarak oluşturuyoruz.

```
public class Iphone6 implements Telefon{
   private String model;
   private String batarya;
   private int en;
   private int boy;

public Iphone6(String model, String batarya, int en, int boy) {
   this.model = model;
   this.batarya = batarya;
   this.en = en;
```

```
    this.boy = boy;
}

@Override
public String getModel() {
    return model;
}

public void setModel(String model) {
    this.model = model;
}
@Override
    public String getBatarya() {
        return batarya;
    }

    public void setBatarya(String batarya) {
        this.batarya = batarya;
    }

    @Override
    public int getEn() {
        return en;
    }

    public void setEn(int en) {
        this.en = en;
    }

    @Override
    public int getBoy() {
        return boy;
    }

    public void setBoy(int boy) {
        this.boy = boy;
    }

    @Override
```

```
    public String toString() {
        return "Iphone6{" +
                "model='" + model + '\'' +
                ", batarya='" + batarya + '\'' +
                ", en=" + en +
                ", boy=" + boy +
                '}';
    }
}
```

Iphone6 sınıfımızı oluşturken implement aldığımız Telefon sınıfındaki kuralları gerçekleştirdik.

Bu noktada elimizde artık telefonumuz var. Şimdi telefonumuzu üreten fabrikayı kodlamamız gerekiyor.Sonra telefonları bayilerde değil bu fabrikalarda üreteceğiz.

```
public class TelefonFabrikasi {
    public static Telefon getTelefon(String model,String batarya,int en,int boy){
        Telefon telefon;
        if ("Iphone6".equalsIgnoreCase(model)){
            telefon=new Iphone6(model, batarya, en, boy);
        }else if ("Iphone8".equalsIgnoreCase(model)){
            telefon=new Iphone8(model, batarya, en, boy);
        }else{
            throw new RuntimeException("Gecerli bir model degildir");
        }


        return telefon;
    }
}
```

Artık getTelefon() metodu ile telefon fabrikasında telefonları üretebiliyoruz. Şimdi telefon bayilerinde istediğimiz telefonları fabrikalardan üretip kullanabiliriz.

```
public class TelefonBayi {
    public static void main(String[] args) {
        Telefon iphone6 = TelefonFabrikasi.getTelefon("Iphone6", "1600mah", 20,
50);
```

```
        System.out.println("iphone6 icin telefon ozellikleri");
        System.out.println(iphone6);

    }
}
```

Böylelikle istediğimizi gerçekleştirmiş oluyoruz.

-Abstract Factory Design Pattern

Yukarıdaki örneğimizde TelefonFabrikasi sınıfında nesneleri oluştururken if kullanıyoruz , bu bizim için sorun teşkil etmiyor gibi gözüksede örneğin 100 model telefon olduğunda tek tek elimizle bu ifleri doldurmamız gerekecek ve buda karmaşaya yol açacaktır. Abstract Factory bu karmaşayı düzeltmek için doğmuştur.

Factory design patern örneğimiz üzerinden bu durumu düzeltmeye çalışalım.

Telefon interface si ile Iphone6 sınıfı aynen kalıyor, iflerden kurtulmak için telefon modellerinide fabrikalarda üretebiliriz yani 1 telefon fabrikasında tüm modelleri üretmek yerine ,o telefon modelini, o telefon modelinin fabrikasında üretebiliriz. Kodlayarak daha iyi anlayalım.

Öncelikle telefon fabrikalarının çatısı olarak TelefonFactory interfacesini oluşturuyoruz.

```
public interface TelefonFactory {
    Telefon getTelefon(String model,String batarya,int en,int boy);
}
```

Şimdi sıra telefon modellerinin fabrikalarını oluşturmaya geldi. Bu fabrikaları oluştururken TelefonFactory interfacesinden implement alıyoruz.

```
public class Iphone6Factory implements TelefonFactory{
    @Override
    public Telefon getTelefon(String model, String batarya, int en, int boy) {
        return new Iphone6(model,batarya,en,boy);
```

```
    }
}
```

Artık her telefon modeli için bir arada defalarca if kullanmamıza gerek kalmadı.

Şimdi sıra telefon bayisi oluşturup fabrikalardan telefon istemeye geldi.

```
public class TelefonBayi {

    public static void main(String[] args) {
        Iphone6Factory iphone6Factory = new Iphone6Factory();
        Telefon iphone6 = iphone6Factory.getTelefon("iphone6", "1600mah", 20,
50);

        System.out.println(iphone6);
    }
}
```

Böylelikle istenilen hedefimize ulaşmış oluyoruz.

-Builder Design Pattern

Bu patternin çıkma amacına değinecek olursak. Kısaca karmaşık nesneleri adım adım oluşturmaya yarar denilebilir. Örneğin bir evlerimizin olduğunu düşünelim bu evlerimizin fazlaca özellikleri olsun, constructor oluşturup bu classdan kullanıcının nesne oluşturmasını istediğimiz zaman kod okunmasını zor hale getiriyor, bir diğer sorun ise constructor da değer girmek istemediğim parametrelere bile değer girilmesi gerekiyor, nesne kararsız bir durumda oluyor, constructor ların bir ismi olmadığı için belirlediğimiz parametreleri alan constructor larımızın isimleri aynı oluyor. Şimdi bu söylediklerimizi nasıl düzelteceğimizi kodlayarak görelim.

Öncelikle Ev classımızı oluşturuyoruz.

```
public class Ev {
    private String il;
    private String ilce;
    private String mahalle;

    private int binaYili;
    private int odaSayisi;
```

```
    private Boolean isDublex;
    private Boolean hasOtopark;

    public Ev() {
    }

    public Ev(String il, String ilce, String mahalle, int binaYili, int odaSayisi,
Boolean isDublex, Boolean hasOtopark) {
        this.il = il;
        this.ilce = ilce;
        this.mahalle = mahalle;
        this.binaYili = binaYili;
        this.odaSayisi = odaSayisi;
        this.isDublex = isDublex;
        this.hasOtopark = hasOtopark;
    }

    public String getIl() {
        return il;
    }

    public void setIl(String il) {
        this.il = il;
    }

    public String getIlce() {
        return ilce;
    }

    public void setIlce(String ilce) {
        this.ilce = ilce;
    }

    public String getMahalle() {
        return mahalle;
    }

    public void setMahalle(String mahalle) {
        this.mahalle = mahalle;
```

```
}

public int getBinaYili() {
   return binaYili;
}

public void setBinaYili(int binaYili) {
   this.binaYili = binaYili;
}

public int getOdaSayisi() {
   return odaSayisi;
}

public void setOdaSayisi(int odaSayisi) {
   this.odaSayisi = odaSayisi;
}

public Boolean getDublex() {
   return isDublex;
}

public void setDublex(Boolean dublex) {
   isDublex = dublex;
}

public Boolean getHasOtopark() {
   return hasOtopark;
}

public void setHasOtopark(Boolean hasOtopark) {
   this.hasOtopark = hasOtopark;
}

@Override
public String toString() {
   return "Ev{" +
         "il='" + il + '\'' +
         ", ilce='" + ilce + '\'' +
```

```
            ", mahalle='" + mahalle + '\'' +
            ", binaYili=" + binaYili +
            ", odaSayisi=" + odaSayisi +
            ", isDublex=" + isDublex +
            ", hasOtopark=" + hasOtopark +
            '}';
    }
}
```

Ev classımız tüm özellikleri içeriyor. Şimdi constructoru bizim için dolduracak EvBuilder classını oluşturuyoruz.

```
public class EvBuilder {
    private String il;
    private String ilce;
    private String mahalle;

    private int binaYili;
    private int odaSayisi;

    private Boolean isDublex;
    private Boolean hasOtopark;

    public static EvBuilder startBuild(){

        return new EvBuilder();
    }

    public Ev builder(){
        Ev ev=new Ev();

        ev.setIl(il);
        ev.setIlce(ilce);
        ev.setMahalle(mahalle);
        ev.setBinaYili(binaYili);
        ev.setOdaSayisi(odaSayisi);
        ev.setDublex(isDublex);
```

```
    ev.setHasOtopark(hasOtopark);

    return ev;
}

public EvBuilder setIl(String il) {
    this.il = il;
    return this;
}

public EvBuilder setIlce(String ilce) {
    this.ilce = ilce;
    return this;
}

public EvBuilder setMahalle(String mahalle) {
    this.mahalle = mahalle;
    return this;
}

public EvBuilder setBinaYili(int binaYili) {
    this.binaYili = binaYili;
    return this;
}

public EvBuilder setOdaSayisi(int odaSayisi) {
    this.odaSayisi = odaSayisi;
    return this;
}

public EvBuilder setDublex(Boolean dublex) {
    isDublex = dublex;
    return this;
}

public EvBuilder setHasOtopark(Boolean hasOtopark) {
    this.hasOtopark = hasOtopark;
    return this;
}
```

```
}
```

Şimdi EvBuilder classında neler yaptık inceleyelim. Öncelikle Ev sınıfımızdaki tüm özellikleri bu sınıfımıza ekliyoruz ardından set metodlarını oluşturuyoruz,

Set metodlarımız normalde void ken biz geriye EvBuilder döndürecek şekilde oluşturuyoruz döndüreceği EvBuilder ise şuanki evi oluşturmalıdır bu yüzden return this yapıyoruz.Böylelikle her seferinde yeni bir özellik ekleyebiliyoruz evimize.

Ardından sınıfımızda ki değerleri setleyebileceğimizbir fonksiyon olan builder() ı oluşturuyoruz. Bu fonksiyonumuz da ev nesnesi oluşturup bu eve clasımızdaki tüm özellikleri setliyoruz. Ardından evi geri döndürüyoruz.

EvBuilder nesnesini oluşturduğumuz startBuild fonksiyonunu yazıyoruz. Bu fonksiyon çağrıldığında EvBuilder nesnemiz oluşmuş oluyor.

Şimdi bir Emlakci sınıfı oluşturup constructor a değer vermeden nasıl yapacağımızı anlatalım.

```
public class Emlakci {
   public static void main(String[] args) {
      Ev ev1 = EvBuilder.startBuild()
            .setIl("Yalova")
            .setIlce("Ciftlikkoy")
            .setBinaYili(2021)
            .setDublex(true).builder();

      print(ev1);
   }

   private static void print(Ev ev1) {
      System.out.println("Ev -->"+ev1);

   }
}
```

StartBuild()fonksiyonunu çağırdığımızda bu fonksiyon bize EvBuilder döndürmektedir. EvBuilderdan set metodlarımızı çağırıp istediğimiz kadar özelliği dolduruyoruz istemediklerimizi doldurmamıza gerek kalmıyor.

Ardından builder metodumuzu çağırdığımızda bu girdiğimiz özelliklerin hepsi ev imizin özelliklerine setlenmiş oluyor bu şekilde isteklerimize ulaşmış oluyoruz.

-Prototype Design Pattern

Bu paternin ortaya çıkma nedeni maaliyeti azaltmak olarak yorumlanabilir. Nedir bu maaliyet, şöyle ki çoğu özelliği aynı olan aralarında küçük farklar olan nesneleri oluşturmamız gerektiğinde sil baştan tüm nesneleri oluşturmamız epey maliyetli olacaktır. Bir diğer kötü yan ise constructor ları her defasında oluşturup bunlara fazlaca olan özellikleri tekrar tekrar girmek yanlış girme risklerine yol açabilmektedir. Gerçek hayattan örnek verecek olursak bir belgemiz olduğunu düşünelim ve bu belgede bir yerde yanlış yaptığımızı varsayalım bu yanlışı düzeltmek için tüm belgeyi baştan yazıp o kısmı düzeltmek yerine belgeyi kopyalayıp o kısmı düzeltebiliriz.

Yazılımda örneklerine bakacak olursak bir oyunumuz olsun ve bunda ev kuralım 1000 tane ev kurduğumuzu düşünelim, bu evler arasında küçük değişiklikler olduğunu var sayarsak bizim maliyetten kaçınmak için bir evden diğerlerini kopyalayıp sadece farklı olan özellikleri eklemeliyiz. Prototype Patternde de tam olarak bu yapılmaktadır nesneler clonlanıp eklenmesi gereken özellikler eklenmektedir.

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

Javada birçok framework bulunmaktadır bunlardan çok kulanılan bazılarına örnek verecek olursak;

-Sipring

-Hibernate

-JavaServerFaces(JSF)

- Struts

-Sipring

Java ile geliştirme yapmayı kolaylaştıran bir frameworktür. Karmaşık ve hantal kurumsal java uygulama geliştirme sürecini basitleştirmeyi amaçlar.Belli web uygulamalarını oluşturma konusunda java kullanan geliştiricilere yardımcı olmak adına tasarlanmıştır. Springin ana avantajlarını düşünecek olursak, açık kaynaklı olmak, önceden hazırlanmış şablonların bulunması, test yapılabilir olması, kullanım kolaylığına sahip olması denebilir.

-Hibernate

Java geliştiricileri için geliştirilmiş ORM kütüphanesidir. Nesne yönelimli oluşturulan modellere göre veritabanı ile olan ilişkiyi sağlar. Veri tabanı üzerinde yapacağımız işlemleri kolaylaştırır. Kurduğumuz yapıyı daha iyi hale getirir. Özetleyecek olursak Javada oluşturduğumuz sınıflardan veri tabanı tabloları oluşturmamamızı sağlar birde java veri tiplerinden SQL veri tiplerine dönüşüm gerçekleştirilmesini sağlar. Bu özellikler sayesinde uygulamaları geliştirirken bizlere kolaylık sağlar işlemleri daha kısa sürede halletmemizi sağlar.

-JavaServerFaces(JSF)

JSF bizlere kullanıcı arayüzleri sağlamaktadır. JSF ile hazır etiketler ile yapmak istediğimiz işlemleri basitçe yapabilmekteyiz. JSF nin yaptığı iş olaylara bağlıdır denebilir burada olaydan kasıt bir butona tıklandığında buna tepki verebilmesidir.

- Struts

Web teknolojilerinde kullanabileceğimiz bir diğer frameworktür. Bu framework MVC mimarisini kullanır. Web tabanlı kurumsal bir proje yaptığımızı düşünelim bu projeyi geliştirmek için projeyi bölümlememiz gerekmektedir. Her ekip kendi alanındaki bölüm ile ilgilenir projenin karmaşık alt tarafları ile ilgilenmez. Ekipdeki değişiklikleri tolare etmek için projede o kısmında uzmanlaşmış birisini bulmak yeterli olacaktır bu kişinin projeyi bilmesine gerek yoktur. Anladığım kadarı ile bu bölümlemeyi bize Struts yapmaktadır

5. Spring frameworkünün kullandığı design patternlar neler?

Factory pattern, Singleton pattern, Prototype pattern, Proxy pattern, Template pattern, Observer pattern, Mediator pattern, Front Controller pattern.

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

-SOA

Temel olarak tüm hizmetlerin farklı birimler tarafından birbirlerinden bağımsız olacak şekilde çalışmasıdır denebilir.

- Web Service

Uzak sistemler veya farklı platformlar arasında XML, JSON gibi ortak bir veri şekli kullanarak veri alışverişini sağlar bunu HTTP protokolü üzerinden yapar.

- Restful Service

Restful Service kavramını anlamak için önce REST in ne olduğunu bilmek gerekiyor. REST client ile server arasındaki haberleşmeyi sağlayan HTTP protokolünün üzerinde çalışmakta olan bir mimaridir. REST servis kullanan mimariler üzerinde oluşturulmuş olan yazılımlarda kullanılan transfer yöntemidir. Istemci ile sunucu arasında haberleşmeyi JSON ve XML verilerini taşıyarak gerçekleştirir.Bu mimariyi yani REST mimarisini kullanan servislere Restful services denir.

-HTTP methods

HTTP, tarayıcı ile web server arasında ki iletişimi kurmamızı sağlayan protokoldür. Bunu sağlarken HTTP metodlarını kullanırlar

Başlıca HTTP metodları şunlardır;

GET, POST, PUT, HEAD, DELETE ,CONNECT, OPTIONS, TRACE, PATCH, SEARCH.