

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

### - Factory Method Design Pattern

Bir üst sınıfta nesneler oluşturmak için bir interface sağlayan, ancak alt sınıfların oluşturulacak nesnelerin türünü değiştirmesine izin veren yaratıcı bir tasarım desendir. Projelerimize bir özellik eklerken en az müdahale ile işlemleri yapmak amaçlanır.

### - Abstract Factory Design Pattern

Factory Design Pattern'da bir ürün ve alt sınıflarına ait bir interface bulunmaktadır. Bu design pattern birbiriyle alakalı veya bağımlı nesnelerin somut sınıflarını belirtmeden, yaratılması için gereken bir interface sağlar.

### - Prototype Design Pattern

Prototip bir nesneyi kullanarak yaratılacak nesneleri belirlemek ve yeni nesneleri kopyalayarak oluşturma amacı ile kullanılır. Özellikle karmaşık ve durumları arasında az farklar olan nesneleri yaratmada kullanılabilir.

### - Builder Design Pattern

Karmaşık olan nesnelerin oluşturulması için kullanılır. Burada karmaşıklıktan kastım çok parametre alan constructorlardır. Nesne oluşturma karmaşıklığı arttığında, nesneyi oluşturmak için başka bir nesne (builder) kullanarak örnekleme sürecini ayrıştırabilir.

### - Singleton Design Pattern

Aslında bu design pattern diğer creational design pattern'ler gibi nesne yaratma ile doğrudan ilgili değildir. Singleton Design Pattern'in temelde odak noktası bir sınıftan sadece bir tane nesne üretme problemidir. Tasarım kalıbı oluşturulan sınıftan sadece bir nesne örneğinin alınmasını sağlar. Bu sayede söz konusu nesneden bir tane yeni bir tane oluşturma veya kopyalama işlemi yapılmadan direkt global olarak kullanılmasını sağlamaktadır.

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

### - Spring

Java uygulamaları geliştirmemizi kolaylaştırır. Spring birçok modülü ve kütüphaneleri kullanmamıza olanak sunar. Springin en temelinde IOC containerı vardır. (Sınıflardan nesne oluşturmak, bağlamak ve yönetme işlemini ele aldığı yer.) Spring XML ve annotation konfigürasyonlarını destekler. Spring singleton ve factory sınıflarının ortadan kaldırılması için gerekli yeteneğe sahiptir. Bağımsız uygulamalar oluşturur.

```
package com.mcnz.rps.spring;

import org.springframework.stereotype.*;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class WebController {

    @GetMapping ("/playagame")
    public String playGame(
        @RequestParam(name="choice", required=false)
        String theChoice,
        Model model) {

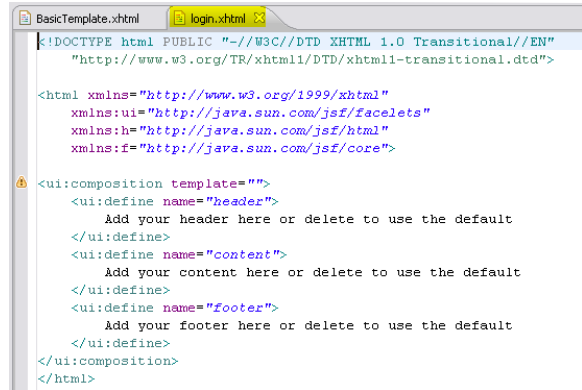
        if (theChoice == null) {
            return "index";
        }

        String theOutcome = "error";
        if (theChoice.equals("rock")) {
            theOutcome = "tie";
        }
        if (theChoice.equals("paper")) {
            theOutcome = "win";
        }
        if (theChoice.equals("scissors")) {
            theOutcome = "loss";
        }

        model.addAttribute("outcome", theOutcome);
        return "results";
    }
}
```

## - JSF (Java Server Faces)

Sunucu tarafında kullanıcı arayüzü oluşturmayı sağlar. Daha çok kurumsal ve zaman kısıtı olan projelerde kullanılabilir. Dil desteği eklenebilir. Validation oluşturulabilir.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

  <ui:composition template="">
    <ui:define name="header">
      Add your header here or delete to use the default
    </ui:define>
    <ui:define name="content">
      Add your content here or delete to use the default
    </ui:define>
    <ui:define name="footer">
      Add your footer here or delete to use the default
    </ui:define>
  </ui:composition>
</html>
```

## - Hibernate

Hibernate bir Orm kütüphanesidir. Nesne yönelimli modellerde veritabanı ile ilişki sağlar ve veritabanı üzerinden yapılan işlemleri kolaylaştırır. Artan veri miktarı Hibernate Framework'ü kullanmanın önemi artmıştır. Daha az kodlama ile verileri işlememize olanak sağlar. Veritabanı bağlantısı kurmak, CRUD işlemlerini gerçekleştirmek gibi şeyler Hibernate tarafından yapılır.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN
3 <hibernate-mapping>
4 <class name="com.jwt.hibernate.Student" table="STUDENT">
5 <id column="ID" name="id" type="long" />
6 <property column="STUDENT_NAME" name="name" type="string" />
7 <property column="DEGREE" name="degree" type="string" />
8 <property column="ROLL" name="roll" type="string" />
9 <property column="PHONE" name="phone" type="string" />
10 </class>
11 </hibernate-mapping>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/javawebtutor</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">mukesh</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="hbm2ddl.auto">create </property>
    <mapping resource="com/jwt/hibernate/student.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

```
1 package com.jwt.hibernate;
2
3 import org.hibernate.Session;
4 import org.hibernate.SessionFactory;
5 import org.hibernate.Transaction;
6 import org.hibernate.cfg.Configuration;
7
8 public class SimpleTest {
9
10     public static void main(String[] args) {
11
12         Configuration cfg = new Configuration();
13         cfg.configure("hibernate.cfg.xml");
14
15         SessionFactory factory = cfg.buildSessionFactory();
16         Session session = factory.openSession();
17         Student student = new Student();
18         student.setName("Mukesh");
19         student.setRoll("101");
20         student.setPhone("8888");
21         student.setDegree("B.E");
22
23         Transaction tx = session.beginTransaction();
24         session.save(student);
25         System.out.println("Object saved successfully.....!!");
26         tx.commit();
27         session.close();
28         factory.close();
29     }
30 }
```

## 5. Spring frameworkünün kullandığı design patternlar neler?

Factory Method, Singleton, Prototype, Proxy, Template Method, Observer, Mediator, Front Controller

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

**SOA (Service Oriented Architecture):** Temelde büyük yazılımların parçalara ayrılıp birbirleriyle network üzerinden iletişim kuran servislerden oluşan bir mimari modeldir. Tek bir amacı gerçekleştirmek için geliştirilen projelerin ilerleyen zamanda ki farklı ihtiyaçlara göre yeniden şekillenmesi ve ek özelliklerinin gelmesi sırasında yaşanacak karışıklığın önüne geçmek amacıyla farklı bir özellik için çalışan takımların network üzerinden iletişim kurarak mimarilerini bu şekilde parçalara ayırmasını örnek gösterebiliriz.

**Web Service:** Web üzerinde http protokolü ile hizmet veren program parçalarıdır. Web servisler, birlikte çalışma, iletişim kurma ve veri alış verişi için standartlaştırılmış web protokolleri sağlayan bir yazılım, uygulama veya bulut teknolojisi içerir. Bir uygulama ve uygulama ile entegre bir şekilde çalışan web sitesinde mesajların hem web sitesine hem de uygulamaya gelmesi örnek olarak gösterilebilir.

**Restful Service:** Rest standartlarına uygun yazılan web servislerine Restful servisler diyoruz. Restful servisler veri iletiminde farklı http metotlarını kullanmaktadır. Bunlar Get, Post, Put ve Delete metotlarıdır.

### HTTP methods :

#### GET

Veri listeleme, veri görüntülemek için kullanılır. GET request'ler güvenli olmalıdır, yani aynı parametrelerle kaç kez tekrar ettiğine bakılmaksızın sonuçlar aynıdır. GET ile veri gönderilirken, adres çubuğunda gönderilir.

Örneğin: Get ile sipariş detaylarını getirebiliriz.

GET : /orders/{id}

#### POST

Veri eklemek için kullanılır ancak mevcut olan bir veriyi güncellemek için de kullanılır. Doğrudan sayfaya veri gönderilir ve veriler adres çubuğunda görünmez. Daha güvenilir yöntemdir.

#### PUT

Veri güncellemek için kullanılır.

Örneğin: PUT ile belirli sipariş güncelleyebiliriz.

PUT : /orders/{id}

#### DELETE

Kaynaktan veriyi silmek için kullanılır.

Örneğin: Id'si verilen adresi silersiniz.

DELETE /addresses/id

#### POST

Örneğin: POST ile yeni bir sipariş oluştururuz.

POST: /orders/