2- Creational Design Patterns, nesnelerin nasıl yaratılacağına belirlemede yardımcı olmak için ortaya çıkmıştır. Objeleri kontrollü bir şekilde oluşturup karmaşıklığı azaltmaya çalışır ve uygulamaya esneklik katar. Esas amaç, objeleri oluştururken bağımsız olarak tasarlamaktır.

Beş farklı Design Pattern bulunmaktadır.

- Abstract Factory
- Factory Method
- Singleton
- Builder
- Prototype

Abstract Factory: Bu yapıda birbirine benzeyen ürünler ortak bir ara katman üzerinden yönetilir bu sayede kurulan yapı daha esnek olur.

Örnek olarak bir mobilya fabrikasını düşünelim.

Bu fabrika **Koltuk**, **Kanepe**, **Sehpa** üretmekte olsun.

Bu ürünleri **Modern** ve **Vintage** üretim tarzı ile üretiyor olsun. Bu mobilyaların kendi aileleriyle eşleşmesi gerekir çünkü müşteriler birbiri ile ilgisi olmayan ürünlerle karşılaşmaktan memnun kalmaz.

Örnek olarak **Vintage** tasarımlı bir **Sandalye** takımı ile **Modern** bir **Kanepe** pek hoş olmaz. Bu nedenle **Abstract Factory** tasarımını burada kullanabiliriz. Her ailenin her ürün tipi için (koltuk, kanepe, sehpa) ayrı bir interface oluşturmamız gerekir. Akabinde tüm varyantların bu interface'i esas almasını sağlayabilirsiniz.

Örneğin tüm koltuk varyantlarında koltuk interface'ini, tüm sehpa varyantlarında sehpa interface'ini esas alabiliriz.

Sonraki adım o aile yapısına ait metotları tanımlamak için gereken **Abstract Factory Interface**'ini tanımlamak, devamında ise **Abstrack Factory Interface**'ini esas alan ayrı ayrı **Factory Class**'ları oluşturmalıyız.

Factory Method: Objelerin nasıl yaratılacağını Inhertiance yoluyla subclass'lara bırakarak obje yaratımı için tek bir Interface kullanarak, Interface üzerinden obje yaratma işlemlerini birbirinden ayırmaya yarayan bir Design Pattern'dür.

Örnek olarak farklı türde araç üretimi yapan bir fabrikamız olsun.

IVehicle adında bir Interface'imiz olsun. Ve bu Interface'imizden Araba, Kamyonet, Tır adında classlar üretiyor olalım.

Devamında ismi VehicleFactory olan içerisinden bize IVehicle ve ProduceVehicle metodu döndüren bir sınıf tanımlayalım. Bu ProduceVehicle metodu VehicleType adında bir Enum parametresi alsın. Bu Enum'u kullanarak metodumuza üretmek istediğimiz tip bilgisini geçeceğiz.

Sonuç olarak yapacağımız değişikliklerden client'ı etkilemeden yapabilmek önceliğimizdir. Ek olarak bir gün Araba değil de XAraba adında IVehicle Interface'ini kullanan bir nesne oluşturabilir ve bunu yaparken client için bir değişikliğe sebep olmaz.

Singleton: Bir class'ın yalnızca bir defa oluşturulmasını öngeren bir tasarım şeklidir. Singleton ayrıca nesneye bir genel erişim noktası sağlar, böylece erişim noktasına yapılan sonraki her çağrı yalnızca o belirli nesneyi döndürür.

Örnek olarak bir devletin sadece bir resmi hükümeti olabilir. Hükümetleri oluşturan bireylerin kişisel kimlikleri ne olursa olsun "AHükümeti" başlığı, sorumlu insan grubunu tanımlayan global bir erişim noktasıdır.

Builder: Tek bir Interface kullanarak karmaşık bir obje grubundan gerektiğince parça yaratılmasını imkân sağlar. Obje oluşturulmak istendiği zaman yapılandırılabilir, bu sayede kullanılmayan parçaların gereksiz yere yaratılarak kaynak harcama durumu ortadan kaldırılmış olur.

Örnek olarak bir Ev class'ı düşünelim. Basit bir ev inşa etmek için dört duvar ve bir zemin inşa etmeniz, bir kapı takmanız, bir çift pencere takmanız ve bir çatı inşa etmeniz gerekir. Mesela ekstra olarak evin ekstra arka bahçesi ya da daha büyük bir salon istiyor isek en basit çözüm olarak temel Ev class'ından extend edip tüm bu parametreleri kapsayacak bir dizi alt sınıf oluşturmaktır.

Çözüm olarak **Builder** tasarım kullanırsak, obje oluşturma kodunu kendi class'ından çıkarmanızı ve onu **Builder** adı verilen ayrı nesnelere taşımanızı önerir.

Prototype: Adından da anlaşılacağı üzere class'lardan obje yaratırken yeni objelerin baştan yaratılmayıp, mevcut class'ları örnek kabul ederek onlardan bir Prototip(kopya) oluşturulmasını sağlar. Bu sayede objeler gereksiz yere kaynak meşgul edilmeden yaratılırlar.

Prototipler kendilerini gerçekten kopyalamazlar. Örnek olarak Mitoz bölünme örneği verilebilir. Orijinal hücre bir prototip görevi görür ve kopyanın oluşturulmasında aktif rol alır.

4- Framework'ler daha başarılı ve kompleks programlar yazmayı daha az satır kod yazarak ve daha kısa sürede ortaya çıkarılabilmeye avantaj sağlar. Aşağıda 3 farklı amaç için kullanılan Java Framework örneklerini görebilirsiniz.

Spring: Spring Framework kurumsal alanda en çok kullanılan Java Framework'üdür. Spring MVC(Model-View-Controller) katmanları kontrol ederek ihtiyacımız olan paket ve sınıfları ekleyip ve bu paketleri kullanabilmemizi sağlayan bir framework'dür.

```
package com.tutorialspoint;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
   }
}
```

```xml
<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">
      <property name = "message" value = "Hello World!"/>
   </bean>

</beans>
```

Java Server Face: JSF dinamik web sayfaları oluşturmamızı sağlayan bir framework'dür. JSF kodları HTML dilinin içine yazılır ve kendine özgü etiketleri vardır. Bu sayede karışıklık olmadan güzel ve düzenli bir performans sunarak web siteler yapmamızı sağlar.

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login</title>
</head>
<body>
<f:view>
    <f:loadBundle basename="de.vogella.jsf.starter.messages" var="msg" />
    <h:form>
        <h:panelGrid columns="2">
            <h:outputLabel value="#{msg.user}"></h:outputLabel>
            <h:inputText value="#{user.name}">
            <f:validator
                validatorId="de.vogella.jsf.starter.validator.LoginValidator" />
            </h:inputText>
            <h:outputLabel value="#{msg.password}"></h:outputLabel>
            <h:inputSecret value="#{user.password}">
            </h:inputSecret>
        </h:panelGrid>
        <h:commandButton action="#{user.login}" value="#{msg.login}"></h:commandButton>
        <h:messages layout="table"></h:messages>
    </h:form>
</f:view>
</body>
</html>
```

Hibernate: Bir ORM kütüphanesidir. Veri tabanı üzerinde yapılan işlemleri kolaylaştırır, Hibernate Java classlarının veri tabanına dönüşümünü yapmaktadır. Aynı zamanda veri çekme ve veri sorgulama işlerinde kullanıcıya oldukça yardım sağlar.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

   <session-factory>
      <property name="hbm2ddl.auto">update</property>
      <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
      <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
      <property name="connection.username">system</property>
      <property name="connection.password">jtp</property>
      <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
   <mapping resource="employee.hbm.xml"/>
   </session-factory>

</hibernate-configuration>
```

```
public class StoreData {
public static void main(String[] args) {

   //Create typesafe ServiceRegistry object
   StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

  Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();
Session session = factory.openSession();
Transaction t = session.beginTransaction();

   Employee e1=new Employee();
   e1.setId(101);
   e1.setFirstName("Gaurav");
   e1.setLastName("Chawla");

   session.save(e1);
   t.commit();
   System.out.println("successfully saved");
   factory.close();
   session.close();

}
}
```

5- **Spring Framework** aşağıdaki **Design Pattern**'leri kullanmaktadır.

- **Factory Method**
- **Singleton**
- **Prototype**
- **Proxy**
- **Template Method**
- **Observer**
- **Mediator**
- **Front Controller**

6-

SOA (Service Oriented Architecture): Servis odaklı mimari, temel olarak her hizmetin farklı birimler tarafından birbirinden bağımsız olarak çalışmasını ifade eder. Birimlerin birbirinden bağımsız olarak çalıştığı bir enviroment'tır.

Örnek olarak kurumsal bir şirketteki İK, IT, Muhasebe vb gibi yapıalrın birbirinden bağımsız ama bir bütün şeklinde çalışması örneğini verebiliriz.

Web Service: HTTP protokolü üzerinden diğer sistemlere veya cihazlara hizmet veren yapılardır.

JSON, XML vb. biçimlerini kullanarak veri alışverişi gerçekleştirirler.

Örneğin, A dili tarafından geliştirilmiş bir programın Y dili ile geliştirilmiş bir mobile program tarafından kullanılmak istenebilir bunun için bu iki tarafın iletişimi için Web Service kullanılır.

Restful Service: REST bir API'ın nasıl çalışması gerektiğine ilişkin koşulları ortaya koyan bir yazılım mimarisidir. Günümüzde web service'ler bu mimariye bağlı olarak yazılmaktadır.

Örneğin, aylık olarak maaş bordroları oluştururken faturalandırmayı otomatik hale getirmek ve dahili bir zaman çizelgesi uygulamasıyla iletişimde olmak için dahili hesap sisteminizin, verileri müşterinin bankacılık sistemiyle paylaşması gerekir. Restful API'ler; güvenli, güvenilir ve verimli yazılım iletişim standartlarını izlediğinden bu bilgi alışverişini destekler.

HTTP Methods: HTTP browser ile server arasındaki iletişim protokülüdür. Aşağıda genel olarak kullanılan HTTP metotlarını görebilirsiniz.

GET: Sunucudan veri almak için kullanılır.

POST: Sunucuya veri yazdırmak için kullanılır.

PUT: Sunucu tarafında bir kaynağı güncellemek için kullanılır.

DELETE: Sunucu tarafındaki herhangi bir beriyi silmeye yarar.

OPTIONS: Sunucu tarafında hangi HTTP metotlarının kullanılabileceğini sorgulamak için kullanılır.