CREATIONAL DESIGN PATTERNS

1. Prototype Design Pattern
2. Builder Design Pattern
3. Singleton Design Pattern
4. Abstract Factory Design Pattern
5. Factory Method Design Pattern

Prototype Design Pattern

Prototype tasarım deseni, Creational Design Pattern grubuna ait olup projemizde kullanacağımız nesneleri new operatörüyle oluşturmanın maliyetini azaltmak için kullanılır. Bir nesneyi birden fazla kullanmak zorunda olduğumuzda nesnenin clone(kopyasını) alırız. Diğer oluşturulacak nesnelerde bu nesnenin prototipi üzerinden üretilir. Üretilen nesnenin çok fazla kaynak tüketmesi engelleyebilmek ve projemizin sınıflardan bağımsız olması gereken yerlerde Prototype kullanabiliriz.

Builder Design Pattern

Builder tasarım deseni, Creational Design Pattern grubuna ait olup basit nesnelerden karmaşık bir nesne oluşturmak için kullanırız. Single responsibility prensibini destekler.Kullanım amacımız kod tekrarından ve performans düşüklüğünden korunmak içindir.

Singleton Design Pattern:

Singleton tasarım deseni, Creational Design Pattern grubuna ait olup projemizde kullanacağımız nesneden sadece bir tane yaratmak istediğimizde kullanırız. Neden Singleton kullanmamız gerekir diye düşünürsek; bir nesnenin yapacağı işlem parametrelere göre değişirse o nesneden birkaç tane yaratmanın anlamı yoktur. Her nesne bellekte yer kaplar ve aynı işleve sahip nesne üretirsek fazla kaynak tüketimine sebep oluruz. Bu yüzden singleton yapma ihtiyacı duyarız. Singleton tasarım kalıbını database işlemlerinde, uygulama ayarları gerektiren işlemlerde ve bağlantı işlemlerinde kullanırız.

Singleton yapmanın bazı kuralları vardır:
Bir sınıftan sadece bir instance alabiliriz. Nesne yaratmak için sınıfın Constructor üyesi üzerinden oluşturur. Sınıfa ait instance ulaşmak için global erişim yapmalıyız. Constructor üyesini private olarak yapıp, new anahtar sözcüğü kullanarak nesne üretilmesini engelleriz. İnstance alabilmemiz için public static bir metot kullanmamız gerekir.

Abstract Factory Design Pattern

Abstract Factory tasarım deseni, Creational Design Pattern grubuna ait olup birden fazla ürün ailesi ile çalışacağımız zamanlarda kullanılır. Birden fazla nesne üretiminde her ürün ailesi için farklı bir arayüz tanımlar. Bu bakımdan factory methodan ayrılır. Nesneye olan bağımlılıkları azaltma amacı vardır. Bu sayede esnek ve geliştirebilir bir yapıya sahip oluruz.

Factory Design Pattern

Factory Design Pattern, uygulamamızda bir superclass'ın nesneler oluşturması için bir interface oluşturmasını sağlar ve subclass nesnelerin tür değiştirmesine izin veren creational design pattern grubuna ait tasarım desenidir. Kalıtım özelliği olan nesnelerin üretilmesi amacıyla kullanılır. Factory method sayesinde var olan nesneleri yeniden oluşturmak yerine yeniden kullanarak sistem kaynaklarını daha verimli bir şekilde kullanabiliriz. Veritabanı bağlantıları, dosya sistemleri ve ağ kaynakları gibi büyük kaynak gerektiren nesneler için kullanabiliriz. Fabrika yöntemleri genellikle Constructor methodlarında private ya da protected erişim belirleyicisi kullanılır.

Önceki ödev üzerine düzenlemeler:

Mesaj.java

```
package Hafta2.Hw2;

import lombok.Setter;

import java.util.Date;
public class Mesaj {

    @Setter
    private String baslik;
    private String icerigi;
    private Date gonderilenTarih;
    private Date okunduguTarihi;
    private boolean gorulduMu;
    private Kullanici gonderici;
    private Kullanici alici;

    public Mesaj(String baslik) {
        super();
        this.baslik = baslik;
    }

}
```

MesajSingleton.java

```
package Hafta2.Hw2;

public class MesajSingleton {

    private static MesajSingleton singleton;

    private MesajSingleton() {

    }

    public static MesajSingleton getSingletonInstance() {
        if (singleton == null) {
            singleton = new MesajSingleton();
            return singleton;
        }
        return singleton;
    }

}
```

MesajRepository.java

```
package Hafta2.Hw2.repository;

import org.springframework.stereotype.Repository;

import Hafta2.Hw2.Mesaj;
import Hafta2.Hw2.Gayrimenkul;

@Repository
public class MesajRepository {

	private static List<Mesaj> mesajKutusu = new ArrayList<();

	static{
		mesajKutusu.add(new Mesaj("baslik1"));
		mesajKutusu.add(new Mesaj("deneme"));
		mesajKutusu.add(new Mesaj("selam"));
	}

	public List<Mesaj> fetchAllMesajlar() {
		return mesajKutusu;
	}

}
```