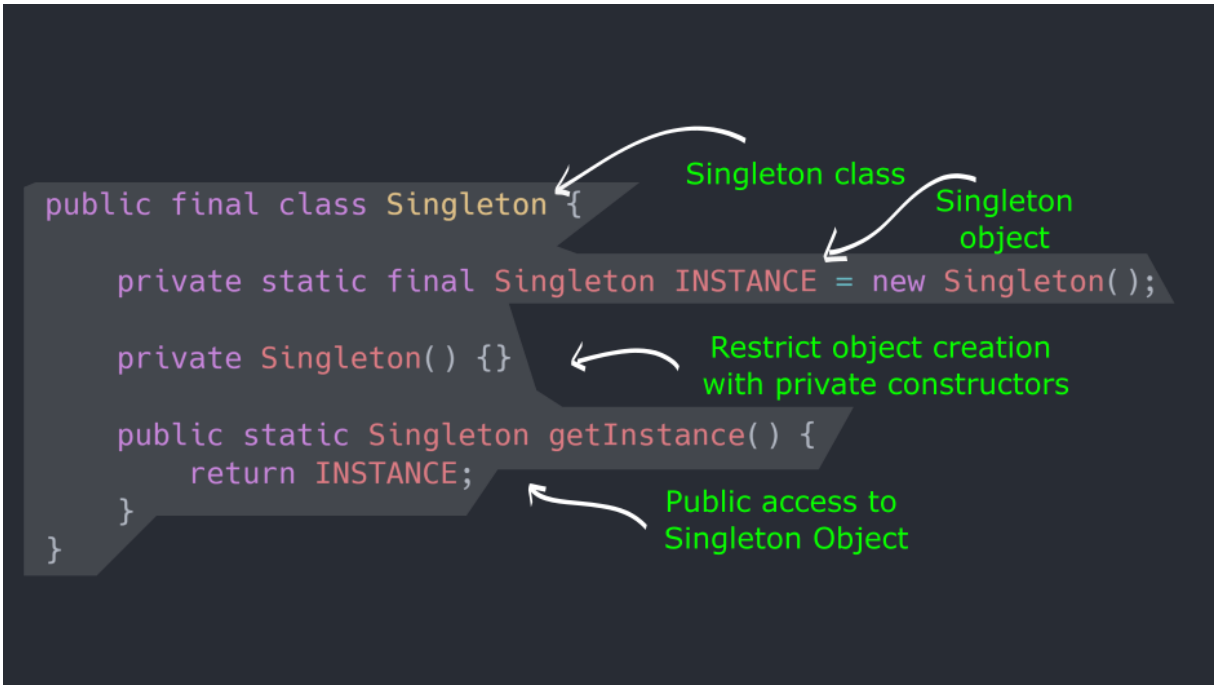
Creational desing paternler ?

1.Singelton:Yanlızca 1 adet yeni obje üretilmesini sağlar .Yani her defasında aynı obje döndürülür.



Yukarıdaki kodda görüldüğü gibi sınıfın default constructor ı private yapılarak dışarıdan erişim engelleniyor .Ve yine private ve static olarak bir insatance oluşturuluyor .Yazılan getinstance metodu ile de her defasında oluşturulan tek obje döndürülüyor.

2.Factory

Bunu doğrudan örnek üzerinden anlatıcam.

```
1   public interface Computer {
2       void name();
3       void since(int year);
4   }
```

Computer.java hosted with ❤ by GitHub

Öncelikle yukarıda computer adında bir interface olduğunu görüyoruz daha sonra bu interfaci implemente eden iki adet sınıf oluşturuyoruz.

```java
public class Mac implements Computer {

    @Override
    public void name() {
        System.out.println("Bilgisayarın Markası Mac");
    }

    @Override
    public void since(int year) {
        System.out.println(year + " senesinde alınmış.");
    }

}
```

```java
public class Asus implements Computer {

    @Override
    public void name() {
        System.out.println("Bilgisayarın Markası Asus");
    }

    @Override
    public void since(int year) {
        System.out.println(year + " senesinde alınmış.");
    }

}
```

Bu sınıfları oluşturduktan sonra bu sınıflardan yeni obje üretmek için aşağıdaki factory sınıfın yazıyoruz

```java
public class ComputerFactory {
    public static Computer createComputer(Class aClass)
            return (Computer) aClass.newInstance();
    }
}
```

ComputerFactory.java hosted with ❤ by GitHub

Bu tasarımda oluşturulacak yeni objelerin türünün ortak olması gerekiyor.

Bunun sağlanabilmesi içinse bu sınıfların ortak bir sınıf veya intrefaceden kalıtım alması gerekli .

3.Builder

Çok fazla alan ve içiçe gemiş nesneler oluştururken ihtiyaç duyacağımız bir tasarım desenidir.Çok fazla alana sahip bir sınıftan obje üretirken çok fazla parametre alan bir constructor çağrısı ve ya çok daha az parametra alana fakat

birden çok constructor ile bu işi yapmamız gerekebilir ve bu iki durumda karmaşıklık doğuracktır.Builder tasarım deseninde ise kurucuyu farklı bir sınıfa dönüştürüp nesne oluşturma işini adım adım yaparak bu karmaşıklığı azaltmayı sağlar .

```
public class Person {

    private String name, surname, address;

    public Person(Builder builder) {
        this.name = builder.name;
        this.surname = builder.surname;
        this.address = builder.address;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public String getAddress() {
        return address;
    }

    public static class Builder{

        private String name, surname, address;

        public Builder(){ }

        public Builder name(String name){
            this.name = name;
            return this;
        }

        public Builder surname(String surname){
            this.surname = surname;
            return this;
        }

        public Builder address(String address){
            this.address = address;
            return this;
        }

        public Company build(){
            return new Company(this);
        }

    }
}
```

Yukarıdaki kodda görüldüğü gibi name surname ve adrees alanlarına sahip person sınıfında nesne oluşturmak için bu üç parametrenin geçildiği bir

constructor yeterli olurdu fakat bu alanların daha fazla ve hangilerinin ilk başta değer alıp almayacağını bilmediğimizi varsayarsak bu durum karmaşık bir hal alır .Bunun yerine bu işlemi inner clas olarak tanımlanan builder clası yardımı ile yapılıryor.

```
Person person = new
Person.Builder().name("Tuğrul").surname("Bayrak").address("Türkiye").
build();
```

Builder sınıfı yardımıyla yuarıda görüldüğü gibi istediğimiz alanlara değer ataması yaparak objeyi oluşturabiliriz.

4.Prototype

Bu tasarım deseni yeni var olan bir objenin aynısını veya çok benzerini oluşturmak istediğimiz zaman ortaya çıkabilecek karmaşayı(çok parametreli constructor)ve bağımlılığı engellemek için kullanılır.Bu yöntem için klon adında bir method içeren bir interface oluştulur.Ve bu interface klonunu oluşturmak istediğimiz nesnenin sınıfına implemente edilir.Bu method sayesinde mevcut sınıfta bir obje oluşur ve bu obje tüm alanlarını eski objeden alır.böylelikle aynı objenin bir kopyası oluşturulur.

```
// Klonlamak istediğimiz sınıflarımıza uygulayacağımız arayüz.
// Farklı sınıflarda da kullanılması için Generic bir yapı oluşturuldu.
// Generic yapı sayesinde dinamik olarak tip dönüşümü sağlanacaktır.
// Bu sayede farklı sınıflara da bu deseni uygulama şansı doğuyor.
interface IClonablePrototype<TPrototype> extends Cloneable {
    TPrototype clone() throws CloneNotSupportedException;
}
```

```
// Employee sınıfına klonlama yeteneği kazandırmak için tanımladığımız
// arayüzü uyguladık.
public class Employee implements IClonablePrototype<Employee> {
    String firstName;
    String lastName;

    public Employee(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public Employee clone() throws CloneNotSupportedException {
        return (Employee) super.clone();
    }
}
```

Yukraıda görüldüğü gibi klon methodun sahip bir interface oluşturarak clonlamak istediğimiz sınıfa extend ediyoruz .Daha sonra bu klon methodunu ovveride ediyoruz .

```
Employee gulizar = new Employee("Gülizar", "Yılmaz");

Employee cloneGulizar = gulizar.clone();


// Aynı nesne örneklerini farklı referanslarda tutuyor.
// Birbirinin aynısı ve farklı referanslarda olan nesneler elde ettik.
System.out.println(gulizar == cloneGulizar); //output: false
```

Daha sonr klon methoduyla yeni bir obje oluşturduğumuzda görüldüğü gibi farklı adreslere sahip aynı objeyi oluşturuyoruz.

5)SPRİNG FRAMEWORKÜN KULLANDIĞI DESİNG PATTERNLER

Factory,singelton ,prototype,Proxy,template ,observer,mediator,front controller desing patternleri kullanılır.