الگوهای طراحی کلاس : GoF

طراحی شی گرای سیستمها

دکتر قاسمی

دانشگاه آزاد تهران شمال

الگوهای طراحی

❖الگوهای طراحی خلاقانهCreational Design Patterns

❖در مهندسی نرم‌افزار، الگوهای طراحی خلقی، الگوهای طراحی هستند که با مکانیسم‌های ایجاد شی سروکار دارند و سعی می‌کنند اشیا را به شیوه‌ای مناسب با موقعیت ایجاد کنند. شکل اصلی ایجاد شی می تواند منجر به مشکلات طراحی یا افزودن پیچیدگی به طراحی شود. الگوهای طراحی خلاقانه این مشکل را با کنترل ایجاد این شیء حل می کنند. الگوهای طراحی خلاقانه از دو ایده غالب تشکیل شده است.

- یکی این است که دانش را در مورد اینکه سیستم از کدام کلاس های الگودار استفاده می کند، محصور می کند.
- دیگری پنهان کردن چگونگی ایجاد و ترکیب نمونه هایی از این کلاس های الگودار است.

انواع الگوها

- الگوهای طراحی خلاقانه بیشتر به الگوهای شی آفرینشی Object-creational و الگوهای کلاس آفرینی Class-creational دسته بندی می شوند،
- الگوهای آفرینشی شی با ایجاد شی و الگوهای خلاقانه کلاس با نمونه سازی کلاس سروکار دارند. در جزئیات بیشتر، الگوهای ایجاد شیء بخشی از ایجاد شیء خود را به شی دیگری موکول می‌کنند، در حالی که الگوهای کلاسی ایجاد ایجاد یا objection خود را به زیر کلاس‌ها موکول می‌کنند.
- پنج الگوی طراحی شناخته شده را معرفی می کنیم که به Gof معروف هستند

۵ الگوی معروف GoF

- متداول ترین الگوهای طراحی خلقی شامل الگوهای Singleton، Factory Method، Abstract Factory، Builderو Prototypeاست. هر یک از این الگوها رویکردهای منحصر به فردی را برای ایجاد اشیا ارائه می دهد، به مشکلات طراحی خاص رسیدگی می کند و مبادلات مختلفی را ارائه می دهد. انتخاب کدام مورد به نیازهای خاص نرم افزار در حال توسعه بستگی دارد.
- *Singleton pattern*، که تضمین می کند یک کلاس فقط یک نمونه دارد و یک نقطه دسترسی جهانی به آن را فراهم می کند.
- *factory pattern* که یک رابط برای ایجاد اشیاء مرتبط یا وابسته بدون تعیین کلاس های عینی اشیاء فراهم می کند.
- *Builder pattern*، که ساخت یک شیء پیچیده را از نمایش آن جدا می کند تا فرآیند ساخت یکسان بتواند نمایش متفاوتی ایجاد کند.
- *Prototype pattern*، که نوع شی را برای ایجاد با استفاده از یک نمونه اولیه مشخص می کند و با شبیه سازی این نمونه اولیه، اشیاء جدیدی ایجاد می کند.

مثال Singleton

```
1  public class Singleton {
2
3   static Singleton theInstance =  new Singleton();
4
5   public int RunCount;
6
7   public static Singleton getInstance() {
8       if (theInstance == null)
9           theInstance = new Singleton();
10
11   return theInstance;
12  }
13
14 }
15
16
17
18 Singleton.getInstance().RunCount++;
```

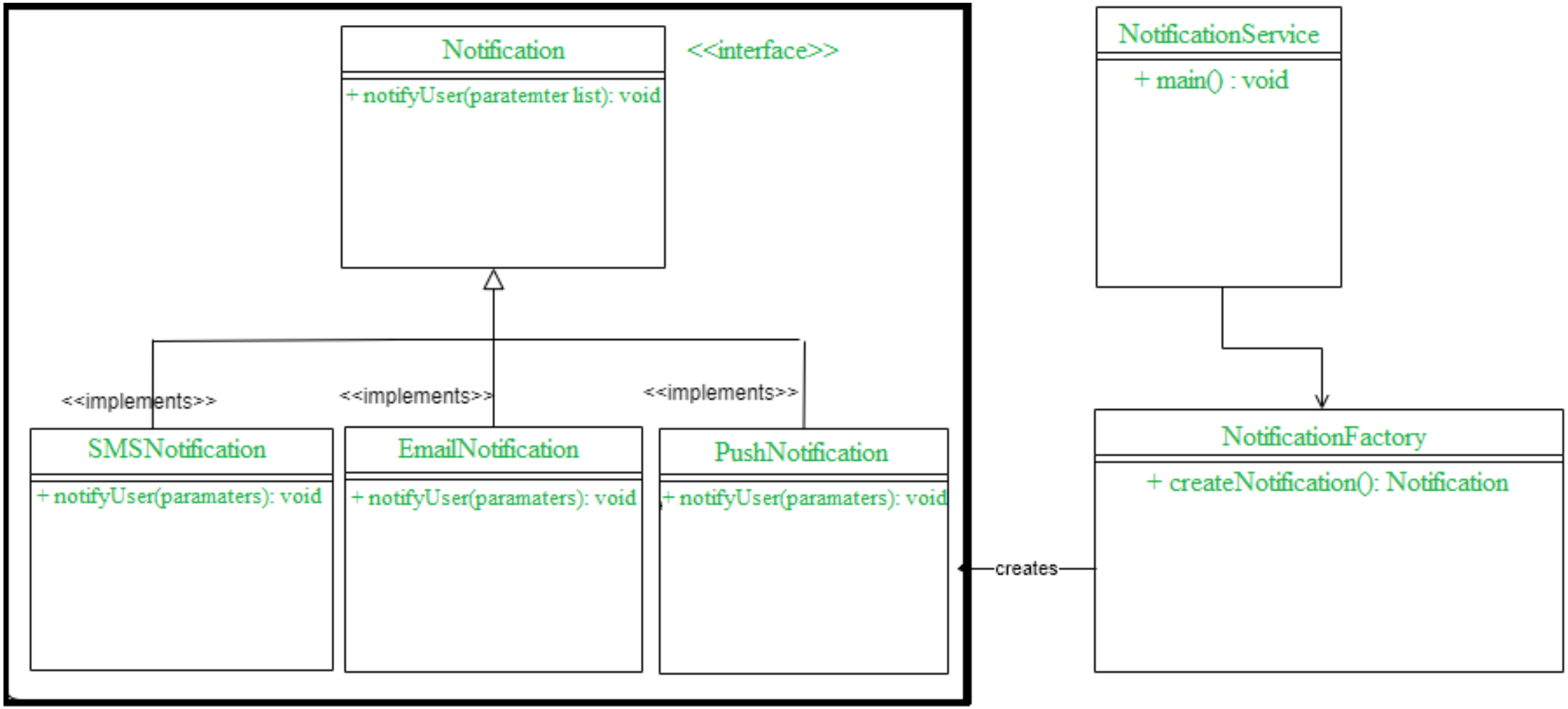
singleton

❖الگوی Singletonتضمین می کند که یک کلاس فقط یک نمونه دارد و یک نقطه دسترسی جهانی به آن کلاس را فراهم می کند. این تضمین می کند که تمام اشیایی که از نمونه ای از این کلاس استفاده می کنند از همان نمونه استفاده می کنند.

❖در زیر مزایای استفاده از الگوی Singletonفهرست شده است:

- دسترسی کنترل شده به نمونه تنها.
- کاهش فضای نام گذاری و فرهنگ لغات سیستم
- اجازه پالایش عملیات و نمایندگی را می دهد.
- تعداد متغیری از نمونه ها را مجاز می کند.
- انعطاف پذیرتر از عملیات کلاس.(راحت تر است)

مثال Factory



مثال Factory

```java
public interface Notification {
    void notifyUser();
}
```

```java
public class SMSNotification implements Notification {

    @Override
    public void notifyUser()
    {
        // TODO Auto-generated method stub
        System.out.println("Sending an SMS notification");
    }
}
```

مثال Factory

```
public class EmailNotification implements Notification {

    @Override
    public void notifyUser()
    {
        // TODO Auto-generated method stub
        System.out.println("Sending an e-mail notification");
    }
}
 public class PushNotification implements Notification {

     @Override
     public void notifyUser()
     {
         // TODO Auto-generated method stub
         System.out.println("Sending a push notification");
     }
 }
```

مثال Factory

```
public class NotificationFactory {
    public Notification createNotification(String channel)
    {
        if (channel == null || channel.isEmpty())
            return null;
        switch (channel) {
        case "SMS":
            return new SMSNotification();
        case "EMAIL":
            return new EmailNotification();
        case "PUSH":
            return new PushNotification();
        default:
            throw new IllegalArgumentException("Unknown channel "+channel);
        }
    }
}
```

الگوی Factory

خصوصیات الگوی Factory

- در یک الگوی Factory، یک کلاس الگودار با متدهای استاتیک برای ایجاد نمونه هایی از اشیاء که یک رابط را پیاده سازی می کنند، استفاده می شود.
- الگوی Factory یک رابط یا interface برای ایجاد یک شی تعریف می کند، اما به کلاس های فرعی اجازه می دهد تصمیم بگیرند که کدام کلاس را نمونه سازی کنند.
- این الگو به شما امکان می‌دهد بدون تغییر کد کلاس‌های جدید را معرفی کنید زیرا کلاس جدید فقط رابط را پیاده‌سازی می‌کند تا مشتری بتواند از آن استفاده کند. شما یک کلاس factory جدید برای ایجاد کلاس جدید ایجاد می کنید و کلاس factory یک رابط یا interface از نوع factory را پیاده سازی می کند.

```java
public class Computer {
    private String HDD;
    private String RAM;
    private boolean isGraphicsCardEnabled;
    private boolean isBluetoothEnabled;
    public String getHDD() {
        return HDD;
    }
    public String getRAM() {
        return RAM;
    }
    public boolean isGraphicsCardEnabled() {
        return isGraphicsCardEnabled;
    }
    public boolean isBluetoothEnabled() {
        return isBluetoothEnabled;
    }
}
```

Builder مثال

```
public class Computer {
    {
    {   private Computer(ComputerBuilder builder) {
        this.HDD=builder.HDD;
        this.RAM=builder.RAM;
        this.isGraphicsCardEnabled=builder.isGraphicsCardEnabled;
        this.isBluetoothEnabled=builder.isBluetoothEnabled;
    }
    //Builder Class
    public static class ComputerBuilder{
        // required parameters
        private String HDD;
        private String RAM;
        // optional parameters
        private boolean isGraphicsCardEnabled;
        private boolean isBluetoothEnabled;
        public ComputerBuilder(String hdd, String ram){
            this.HDD=hdd;
            this.RAM=ram;
        }
        public ComputerBuilder setGraphicsCardEnabled(boolean isGraphicsCardEnabled) {
            this.isGraphicsCardEnabled = isGraphicsCardEnabled;
            return this;
        }
        public ComputerBuilder setBluetoothEnabled(boolean isBluetoothEnabled) {
            this.isBluetoothEnabled = isBluetoothEnabled;
            return this;
        }
        public Computer build(){
            return new Computer(this);
        }
    }}
```

مثال Builder

```
public class TestBuilderPattern {

    public static void main(String[] args) {
        //Using builder to get the object in a single line of code and
                //without any inconsistent state or arguments management issues
        Computer comp = new Computer.ComputerBuilder(
                "500 GB", "2 GB").setBluetoothEnabled(true)
                .setGraphicsCardEnabled(true).build();
    }

}
```

ویژگی های الگوی Builder

❖در حوزه الگوهای طراحی نرم‌افزار، مقوله ایجادی عمدتاً به مکانیسم‌های ایجاد شی مربوط می‌شود، تلاش می‌کند تا اشیاء را به شیوه‌ای مناسب با موقعیت ایجاد کند و در عین حال سیستم را نسبت به ویژگی‌های اشیاء در حال ایجاد ناشناس نگه می‌دارد.

❖در میان این الگوها، الگوی سازنده ابزاری است که به طور منحصربه‌فردی به چالش‌های ارائه‌شده توسط اشیایی که نیازمند فرآیندهای ساخت‌وساز پیچیده هستند، می‌پردازد.

❖مزایای Builder :

- اگر تعداد فیلدهای مورد نیاز برای ایجاد یک شی بیشتر از ۴ یا ۵ باشد، کد قابل نگهداری تر است.
- کد کمتر احتمال دارد که خطا ایجاد کند زیرا کاربر می داند که به دلیل فراخوانی روش صریح چه چیزی ارسال می شود.
- کد قوی تر است زیرا فقط شی کاملاً ساخته شده در دسترس مشتری خواهد بود.

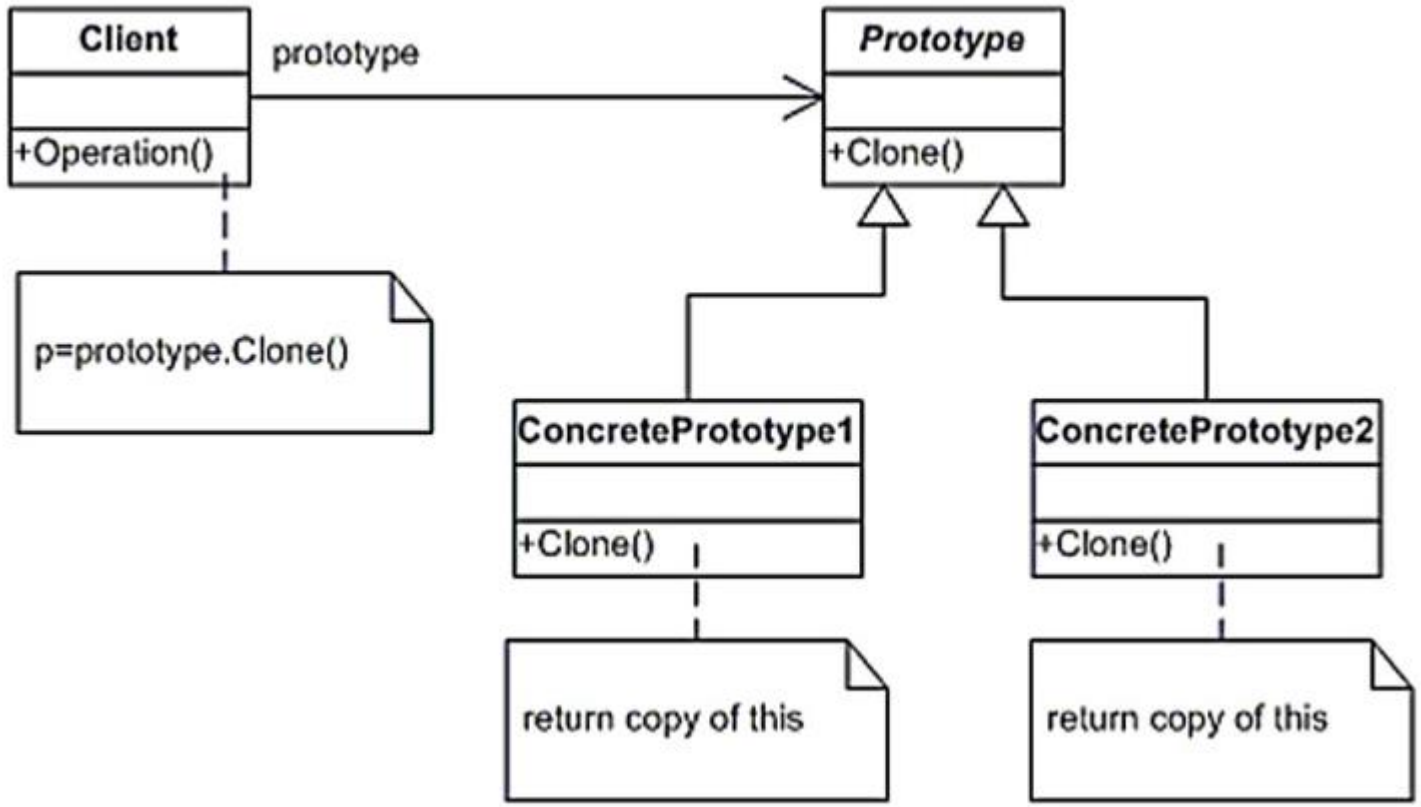
الگوی Prototype

❖همان طور که در الگوی Factory دیدیم، آن‌ها به کلاینت اجازه می‌دادند که به فرایند ایجاد یک شی وابسته نباشد؛ به عبارتی این الگو اجازه می‌دهد تا کلاینت با استفاده از یک متد، کلاس مناسب خود را داشته باشد، بدون اینکه به طور دقیق کلاس مناسب خود را ذکر کرده باشد.

❖اما مشکل هر دو الگو زمانی است که ما بخواهیم یک مجموعه از اشیا شبیه به هم یا مختلف داشته باشیم؛ در این حالت باید به ازای هر کدام از آن‌ها یک Factoryداشته باشیم، آن هم وقتی که هزینه ساخت اشیا از کلاس‌ها زیاد است.

❖در صورتی که الگوی Prototype انعطاف‌پذیرتر است. ایده اصلی این الگو به این صورت است که برای ساخت نمونه از اشیا، به جای استفاده از دستور New در بخش‌های مختلف برنامه، از اولین شی ایجاد شده کپی‌برداری کنید؛ این کار باعث صرفه‌جویی در زمان و فضای استفاده شده برای برنامه می‌شود.

الگوی prototype



نمونه الگوی prototype با متد clone

```
public abstract class Shape
{
    public String id;
    protected String type;
    abstract void draw();

    public Shape clone()
    {
        Shape clone = new Shape();
        try
        {
            clone.type= this.type;
            clone.id= this.id++;
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return clone;
    }
}
```

مزایای استفاده از الگوهای طراحی

- باعث خوانا تر شدن برنامه شما می شود و همه با این الگوها اشنا هستند
- باعث استاندارد شدن کد شما می شود و توسعه دهندگان دیگر متوجه نحوه استفاده از کدهای شما در برنامه های خود می شوند
- باعث ساده‌سازی و تسریع روند نمونه‌سازی از کلاس‌های بزرگ و سنگین می شود.
- به کلاینت اجازه استفاده از کلاس‌های مشخصی از برنامه را بدون هیچ اصلاح و تغییری می‌دهد.


Q&A