

**دانشگاه گیلان**

**دانشکده فنی**

**گروه مهندسی کامپیوتر**

**الگوهای طراحی (Design Patterns)**

(گزارش‌کار آزمایشگاه مهندسی نرم‌افزار)

**اعضای گروه:** مهران رفیعی - پارسا عباسی

**شماره دانشجویی:** 940122610025 - 94012269006

**تاریخ ارائه:** 26 آبان 97

**استاد مربوطه:** مهندس فرید فیضی

**محل ارائه:** آزمایشگاه نرم‌افزار – دانشکده فنی – دانشگاه گیلان

**فهرست مطالب**

چکیده 5

فصل اول: مقدمه 10

1-1- تاریخچه و تعریف 10

2-1- اجزای الگو‌های طراحی 10

3-1- الگو‌های طراحی در مهندسی نرم‌افزار 10

4-1- انواع الگو‌های طراحی 10

1-4-1- انواع الگو‌های طراحی: براساس هدف 10

2-4-1- انواع الگو‌های طراحی: براساس حوزه 10

فصل دوم: Memento pattern

1-2- تعریف 10

2-2- مثال 10

3-2- پیاده‌سازی 10

4-2- کد 10

فصل سوم: Builder pattern

3-1- تعریف 11

3-2 مثال 11

3-3 پیاده‌سازی 12

فصل چهارم: Adapter pattern

4-1 تعریف 13

4-2 مثال 13

4-3 پیاده‌سازی 14

فصل پنجم: Observable pattern

5-1 تعریف 15

5-2 مثال 15

5-3 پیاده‌سازی 15

مراجع 16

**چکیده**

الگو‌های طراحی[[1]](#footnote-1)، در حوزه‌ی مهندسی نرم‌افزار[[2]](#footnote-2)، راه حل های عمومی و تعمیم یافته ای هستند که برای یک مسئله‌ی رایج در طراحی نرم‌افزار به کار می روند. در واقع این الگو‌ها، ساختار راه حل هایی را ارائه می دهند که برای یک مساله و مانند آنها، به عنوان راه حلی رایج و استاندارد پذیرفته شده است. در این ارائه به تعریف الگو‌های طراحی و همچنین تشریح برخی از الگو‌های نام آشنا پرداخته خواهد شد.

**فصل اول  
مقدمه**

**1-1- تاریخچه و تعریف**

ایده‌ی الگو‌ها برای نخستین بار توسط کریستوفر الکساندر[[3]](#footnote-3)، یک معمار، مطرح و پا به حوزه های تخصصی گذاشت. تعریف او از الگو‌ها بدین صورت بود که «هر الگو، مسئله ای را توصیف می کند که بارها و بارها در محیط ما تکرار می شود و سپس به توصیف مرکز راه حل آن می پردازد، بدین صورت که از این راه حل می توان میلیون ها بار استفاده کرد، بدون آنکه حتی دوبار به طریق مشابه عمل کرد.» ]1[

**2-1- اجزای الگو**

هر الگو، دارای 4 جز اصلی است :

* نام الگو
* مسئله
* راه حل
* عواقب

**3-1- الگو‌های طراحی در مهندسی نرم‌افزار**

در حوزه‌ی مهندسی نرم‌افزار، برای نخستین بار، گروهی متشکل از 4 نویسنده ملقب به GoF[[4]](#footnote-4)، دست به انتقال مفاهیم الگو‌های طراحی به این حوزه زدند و با تالیف کتابی به همین نام[[5]](#footnote-5)، جهان حل مسئله در طراحی نرم‌افزار را دگرگون کردند. این کتاب، تاکنون نیز به عنوان اصلی ترین مرجع الگو‌های طراحی در مهندسی نرم‌افزار به شمار می رود و به بیش از 13 زبان ترجمه شده است.

به تعریف این کتاب، الگو طراحی در حوزه مهندسی نرم‌افزار عبارت است از «یک راه حل عمومی قابل تکرار، در پاسخ به یک مسئله‌ی رایج هنگام طراحی نرم‌افزار». این بدین معنی است که یک الگو طراحی، یک طراحی تمام و کمال نیست که بتواند مستقیماً به کد تبدیل شود، بلکه توصیف یا قالبی برای چگونگی حل مسئله است که بتواند در موقعیت های مختلفی مورد استفاده قرار گیرد.

الگو‌های طراحی، بِه روش هایی[[6]](#footnote-6) را ارائه می کند که توسط توسعه دهندگان با تجربه در برنامه نویسی شی گرا، تجربه و مورد استقبال قرار گرفته است. الگو‌های طراحی، راه حل هایی هستند برای مسائلی که اکثر توسعه دهندگان در هنگام توسعه نرم‌افزار خود با آنها روبرو می شوند.

شایان ذکر است که در اینجا الگو‌های طراحی، در حوزه‌ی برنامه نویس شیء گرا و همراه با مفاهیم کلاس، اشیاء، و... بحث می شوند. بنابراین باید به دو نکته توجه داشت :

* الگو‌های طراحی در مهندسی نرم‌افزار، به معنی طراحی ها و ساختمان داده ها نیستند. به عنوان مثال، لیست پیوندی یا جدول هش یک الگو طراحی به حساب نمی آیند، بلکه آنها در کلاس هایی کدگذاری شده و قابل استفاده مجدد هستند.
* الگو‌های طراحی در مهندسی نرم‌افزار، توصیف هایی از ارتباط بین اشیاء و کلاس ها هستند که برای حل یک مسئله‌ی رایج در طراحی نرم‌افزار، سفارشی سازی شده اند.

**4-1- انواع الگو‌های طراحی**

ما الگو‌های طراحی را به دو شاخص تقسیم می کنیم : ]2[

* نخستین شاخص، «هدف»[[7]](#footnote-7) نام دارد و آنچه را که الگو انجام می دهد، منعکس می کند.
* شاخص دوم، «حوزه»[[8]](#footnote-8) نامیده می شود و مشخص می کند که الگو در درجه اول بر روی کلاس ها اعمال می شود یا شیء ها.

**1-4-1- انواع الگو‌ها: براساس هدف**

الگو‌ها براساس هدف به سه دسته‌ی زیر تقسیم می شوند :

* خلاقانه[[9]](#footnote-9) :  
  الگو‌های خلاقانه، به فرآیند تولید اشیاء، پیکربندی و مقداردهی اولیه کلاس ها و اشیاء توجه دارند.
* ساختاری[[10]](#footnote-10) :   
  الگو‌های طراحی با ترکیب کلاس ها و شیء ها، سر و کار دارند.
* رفتاری[[11]](#footnote-11) :  
  الگو‌های رفتاری، روش هایی را که کلاس ها یا شیء ها برای تعامل و توزیع مسئولیت ها در پیش می گیرند، توصیف می کنند.

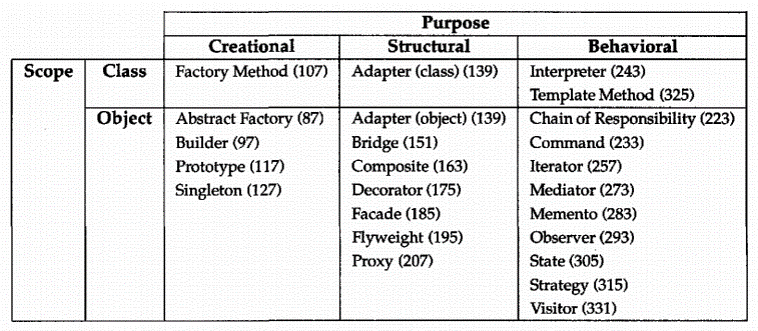
**2-4-1- انواع الگو‌ها: براساس حوزه**

الگو‌ها براساس حوزه به دو دسته کلی تقسیم می شوند :

* کلاس[[12]](#footnote-12) :  
  اینگونه الگو‌ها، با ارتباط کلاس ها و زیرکلاس ها، سر و کار دارند. باید توجه داشت که این ارتباط ها در طول عمل وراثت ایجاد می شوند، بنابراین بصورت ایستا (ثابت در زمان کامپایل) هستند.
* شیء[[13]](#footnote-13) :  
  این الگو‌ها با ارتباط اشیاء سر و کار دارند، و از آنجایی که اشیاء می توانند در طول زمان اجرا تغییر کنند، الگو‌هایی پویاتر هستند.

در جدول 1-1 می توان لیست الگو‌های پیشنهادی و تشریح شده در کتاب مرجع ]2[ را همراه با دسته بندی آنها ملاحظه کرد. هر کدام این الگو‌ها امروزه نیز به عنوان راه حل هایی استاندارد و رایج به شمار می روند.

(جدول 1-1)



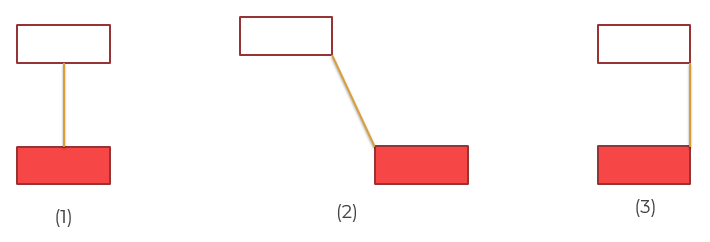
**فصل دوم  
pattern Memento**

**1-2- تعریف**

لغت [[14]](#footnote-14)Memento به معنی شیء است که برای یادآوری یک شخص یا رخداد نگهداری می شود. ]3[  
در مهندسی نرم‌افزار، Memento برای بازیابی حالت یک شیء به حالت پیشین آن به کار می رود. ]4[  
قابلیت بازگشت به عقب[[15]](#footnote-15) موجود در اکثر برنامه های کاربردی، نمونه ای پرکاربرد از همین الگو است.  
شایان ذکر است این الگو در مجموعه الگو‌های رفتاری می گنجد، از همین رو از پویایی مناسبی برخوردار است.

**2-2- مثال**

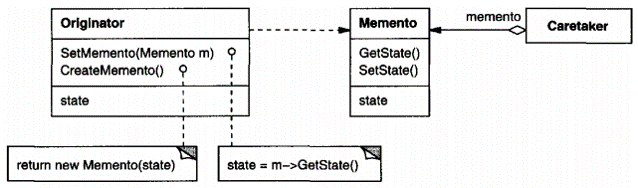
تصور کنید در یک برنامه‌ی رسم مثل رسم فلوچارت، دو مستطیل زیر را با یک خط به یکدیگر متصل کرده اید (حالت 1). حال اگر مستطیل پایینی را به سمت راست بکشانید، انتظار می رود که خط اتصال بین دو مستطیل نیز حفظ شود (حالت 2). از همین رو از یک تابع ریاضی برای پیدا کردن نقاط شروع و پایان این خط اتصال کمک گرفته می شود. حال فرض کنید که قصد بازگشت به حالت قبلی را داشته باشید. از آنجایی که تابع ریاضی، صرفاً به برقراری اتصال خط با استفاده از یک فرمول ساده می پردازد و هیچ اطلاعی از محل قبلی این خط نداشته، آن را در سمت راست (اولین فاصله‌ی کوتاه) رسم می کند (حالت 3). بنابراین نیاز است که ما اطلاعاتی از قبیل نقطه‌ی شروع خط اتصال و نقطه‌ی پایان آن را بر روی مستطیل ها ذخیره کنیم تا در صورت نیاز به بازیابی آنها بپردازیم. این ذخیره سازی بر روی حالت ها و به کمک الگو Memento انجام می پذیرد. ]2[



(شکل 2-1)

**3-2- پیاده سازی**

ساختار این الگو به صورت زیر است ]2[ :



(شکل 2-2)

* Caretaker مسئول بازیابی حالت شیء از روی Memento است.
* Memento شامل حالتی از یک شیء است که می تواند بازیابی شود.
* Originator به ساخت و ذخیره‌ی حالت ها در شیء یادآور می پردازد.

**4-2- کد**

public class Originator {

private String state;

public void setState(String state){

this.state = state;

}

public String getState(){

return state;

}

public Memento saveStateToMemento(){

return new Memento(state);

}

public void getStateFromMemento(Memento memento){

state = memento.getState();

}

}

public class Memento {

private String state;

public Memento(String state){

this.state = state;

}

public String getState(){

return state;

}

}

import java.util.ArrayList;

import java.util.List;

public class CareTaker {

private List<Memento> mementoList = new ArrayList<Memento>();

public void add(Memento state){

mementoList.add(state);

}

public Memento get(int index){

return mementoList.get(index);

}

}

**فصل سوم  
Builder pattern**

**1-3- تعریف**

[7]لغت Builder به معنای سازنده است، همانطور از معنای لغت پیداست این الگو هنگام ساختن یک کلاس یا درواقع تعریف یک کلاس استفاده میشود. به عبارت دیگر با استفاده از این الگو هنگام ساختن یک شی از یک کلاس میتوان علاوه بر استفاده راحت تر، در یک چهارچوب تعریف شده از این کلاس استفاده نمود. اما این‌ها به چه معنی است؟چگونه این الگو ساختن یک شی از یک کلاس را راحت‌تر می‌کند؟

**2-3- مثال**

اجازه بدهید با یک مثال قضیه را روشن کنیم:

فرض کنید یک کلاس Car تعریف کرده‌ایم. این کلاس دارای پارامترهایی نظیر:

* تعداد در های ماشین
* رنگ
* کارخانه سازنده
* مدل
* نوع ماشین (کوپه – SUV - ..)
* ...

میتواند باشد. در این مثال تعداد پارامتر‌های کلاس کم است، اما فرض کنید کلاس ما 100 پارامتر داشته باشد!

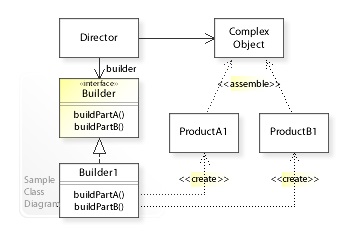
حال اگر کسی بخواهد یک شی از این کلاس بسازد باید همه این 100 پارامتر را یا در constructor و یا توسط توابع get و set این کلاس مشخص کند. مسلما کدی که نوشته میشود بسیار کثیف و شلوغ خواهد بود. برای رفع این مشکل الگوی Builder یا سازنده روشی را پیشنهاد میدهد که روش‌های پیاده‌سازی متفاوتی دارد، اما بگذارید یک روش پیاده‌سازی را توضیح دهیم.

یکی از روش‌های پیاده‌سازی این الگو بدین صورت است که ابتدا یک Interface برای کلاس اصلی (Car) تعریف کنیم. حال فرض کنید همه ماشین هایی که قرار است ساخته شوند از کارخانه BMW است، بجای اینکه به ازای هر شی‌ای که میسازیم به پارامتر کارخانه آن مقدار BMW بدهیم میتوانیم یک کلاس دیگر ساخته تا از interface مرحله قبل استفاده کند.

این کلاس جدید بطور پیش فرض مقدار BMW را در دل خود دارد و هنگام ساختن شی جدید به راحتی از همین کلاس میتوان استفاده نمود.

یکی دیگر از مزیت‌های این روش این است که میتوانیم کلاس را طوری طراحی کنیم که شخصی که میخواهد شی جدید از نوع BMW را بسازد نتواند پارامترکارخانه را تغییر دهد.

**3-3- پیاده سازی**

****

شکل 2-1

در شکل 2-1 واضح است کهDirector بجای آنکه مستقیما از Complex object استفاده کند بصورت غیرمستقیم از interface آن استفاده میکند و بقیه کارها را به دست interface میسپارد.\*

\* : برای مشاهده نمونه پیاده‌سازی شده در جاوا به لینک روبرو مراجعه کنید: <https://github.com/mehran75/DesignPatterns/tree/master/src/builder_pattern>

**فصل چهارم**

**Adapter pattern**

**1-4- تعریف**

در لغت Adapter به معنای وسیله‌ای است که دو وسیله دیگر که قابل اتصال نیستند را به یکدیگر متصل کند. مثلا شارژر گوشی‌موبایل برق 220 ولت شهر را تبدیل به 5 ولت میکند تا قطعات گوشی منفجر نشوند!

در برنامه نویسی نیز دقیقا از همین تعریف میتوان استفاده کرد، درواقع دو یا چند ماژول که قابل اتصال مستقیم را ندارند به یکدیگر مربوط میکند. اما چطور؟!

**2-4- مثال**

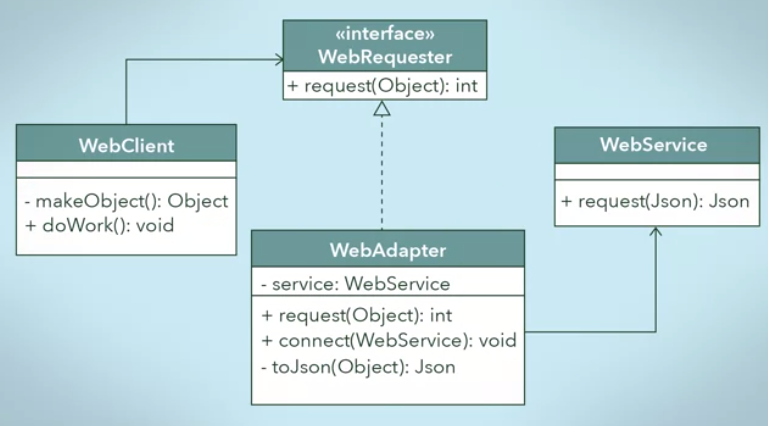
[6]فرض کنید که ما یک وب سرویس پیاده‌سازی کرده ایم که یک رشته بعوان ورودی گرفته و پاسخ متناظر آنرا تولید میکند. حال فرض کنید کلاینتی داریم که برای درخواست از وب سرویس‌هایی پیاده‌سازی شده اند که ورودیشان object است!

مشکل اینجاست که نباید به کد هیچکدام از این دو دست بزنیم، و باید بدون اعمال هیچ تغییری در ورودی و خروجی این دو را به یکدیگر متصل کنیم.

مسلما این دو کامپوننت قادر به همکاری نیستند مگر اینکه یک اداپتور بعوان واسط بین این دو عمل کند.

کل کاری که این اداپتر قرار است انجام دهد تبدیل object به String و پاسخ دریافت شده از وب سرویس را به مشتری اعلام کند.

**3-4- پیاده‌سازی**



شکل 4-1

شکل بالا دقیقا نحوه پیاده‌سازی مثال قبل را نشان میدهد.

برای دسترسی به پیاده‌سازی این مثال به زبان جاوا میتوانید از لینک زیر استفاده کنید:

<https://github.com/mehran75/DesignPatterns/tree/master/src/adapter>

**فصل پنجم**

**Observer pattern**

**1-5- تعریف**

طبق معمول ابتدا به معنی لغوی این الگو میپردازیم. معنی این الگو مشاهده‌گر است. این اسم دقیقا با کاری که قرار است انجام دهد مطابقت دارد، یعنی مشاهده کند!

فرض کنید یک فرد معروف در صفحه اینستاگرام خود عکسی از خود به اشتراک میگذارد. همه طرفداران او منتظر عکس جدید او هستند (observer) و به محض آپلود شدن عکس توسط فرد معروف (Observable) خبردار میشوند و نیازی نیست فرد معروف به تک تک آنها اطلاع دهد.

دربرنامه نویس نیز به همین صورت است. به مثال قسمت بعد توجه کنید.

**2-5- مثال**

برای مثال ما یک progress bar برای نمایش میزان سنجش آپلود یک فایل در برنامه گذاشته ایم.

برای آپدیت کردن این progress bar دو راه وجود دارد:

1. هربار مقدار پیشرفت را مستقیما به progress bar اعلام کنیم.
2. از یک شی Observable استفاده کنیم.

میدانیم که برای آپلود فایل باید یک thread جدا از گرافیک برنامه استفاده کنیم تا هنگام آپلود برنامه به اصطلاح freeze نشود. در جاوا (JavaFX) این روش بصورت زیر باید صورت گیرد:

Platform.runLater( ()-> {

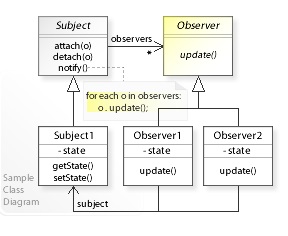
Progressbar.setValue(value);

});

این روش علاوه بر شلوغ کردن کد، بسیار غیربهینه میتواند باشد.

اما پیاده سازی روش دوم چگونه است؟ در بخش بعدی توضیح روش دوم را مشاهده کنید.

**3-5- پیاده‌سازی**

****

شکل 5-1

در شکل بالا مشاهده میکنید که بجای روش اولی که توضیح دادیم کافیست یک واسط بین progress bar و thread مربوط به آپلود فایل قرار دهیم و بجای استفاده از کد قبل از این کد استفاده میکنیم:

SimpleDoubleProperty observable = new SimpleDoubleProperty(0.0);

------------------------------------ in upload thread ----------------------------------

observable.set(value);

------------------------------------ in GUI thread -------------------------------------

progressbar.valueProperty().bind(observable);

مشخص است که کد بالا بسیار خواناتر و تمیز تر است.

برای دسترسی به پیاده‌سازی این مثال به زبان جاوا میتوانید از لینک زیر استفاده کنید:

<https://github.com/mehran75/DesignPatterns/tree/master/src/observable>

**مراجع**

[1] A Pattern Language: Towns, Buildings, Construction  
 by Christopher Alexander, Sara Ishikawa, Murray Silverstein

[2] Design Patterns: Elements of Reusable Object-Oriented Software  
 by Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm

[3] Oxford Online Dictionary

[4] Design Patterns - Memento Pattern  
 [https://www.tutorialspoint.com/design\_pattern/memento\_pattern.htm]

[5] Memento Design Pattern  
 [https://javapapers.com/design-patterns/memento-design-pattern/]

[6] coursera

[7] tutorialspoint website

1. Design Patterns [↑](#footnote-ref-1)
2. Software Engineering [↑](#footnote-ref-2)
3. Christopher Alexander [↑](#footnote-ref-3)
4. Gang of Four [↑](#footnote-ref-4)
5. Design Patterns: Elements of Reusable Object-Oriented Software [↑](#footnote-ref-5)
6. Best Practices [↑](#footnote-ref-6)
7. Purpose [↑](#footnote-ref-7)
8. Scope [↑](#footnote-ref-8)
9. Creational [↑](#footnote-ref-9)
10. Structural [↑](#footnote-ref-10)
11. Behavioral [↑](#footnote-ref-11)
12. Class [↑](#footnote-ref-12)
13. Object [↑](#footnote-ref-13)
14. نشان، یادگاری [↑](#footnote-ref-14)
15. Undo [↑](#footnote-ref-15)