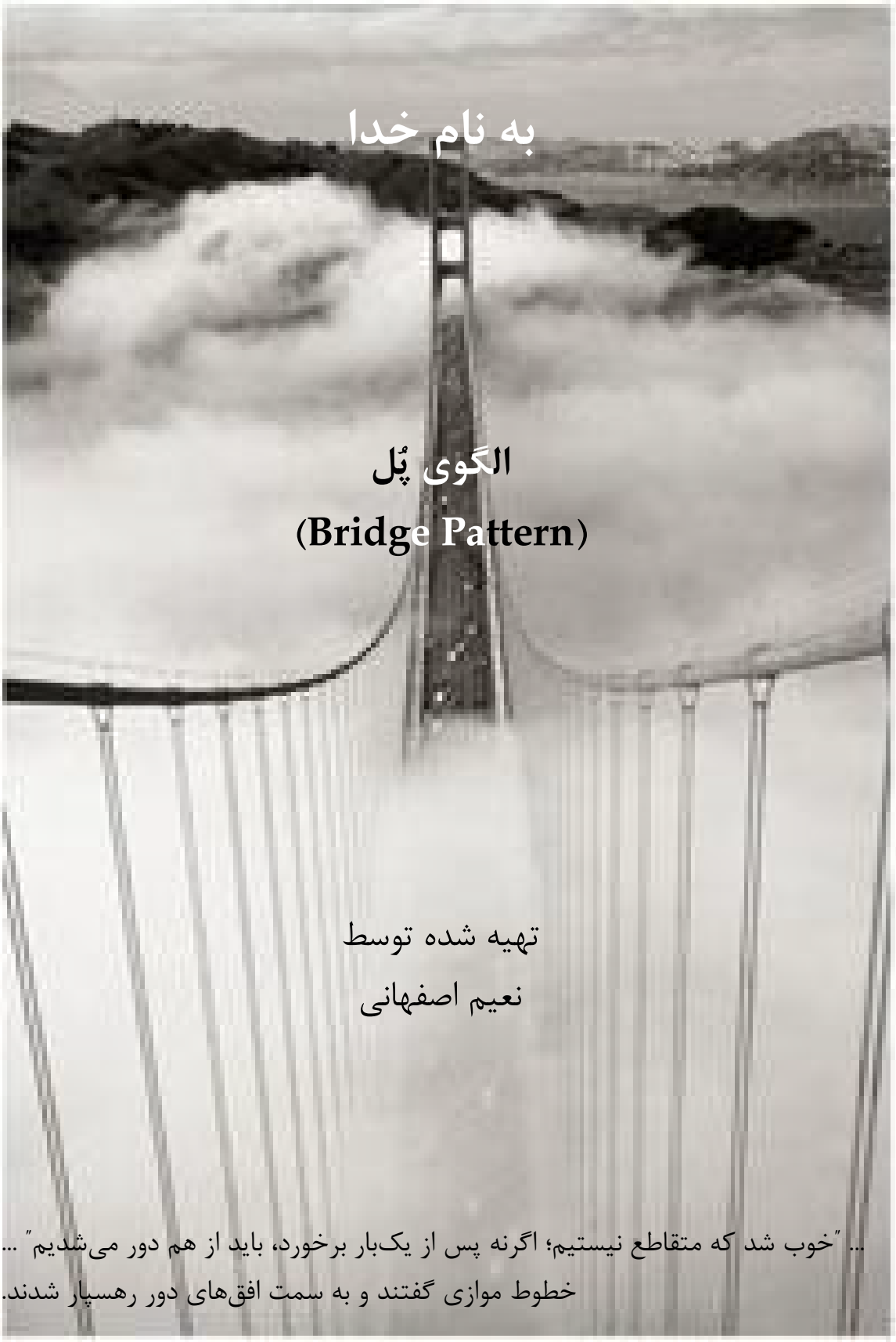


به نام خدا

الگوی پُل
(Bridge Pattern)

تهیه شده توسط
نعیم اصفهانی

... "خوب شد که متقاطع نیستیم؛ اگرنه پس از یک‌بار برخورد، باید از هم دور می‌شدیم" ...
خطوط موازی گفتند و به سمت افق‌های دور رهسپار شدند.

تعریف:

تجریدها[1] را از پیاده‌سازی‌ها[2] جدا کنیم تا هر دو بتوانند به‌طور مستقل تغییر کنند.

تجرید به این معناست که اشیاء مختلف چگونه از نظر منطقی به هم مرتبطند.

پیاده‌سازی در این‌جا به معنای اشیائی است که کلاس مجرد و پیاده‌سازی‌هایش برای پیاده‌سازی خود آن‌ها را استفاده می‌کنند. نه پیاده‌سازی‌های کلاس مجرد که کلاس واقعی نامیده می‌شوند.

دو قانون کلی در الگوها:

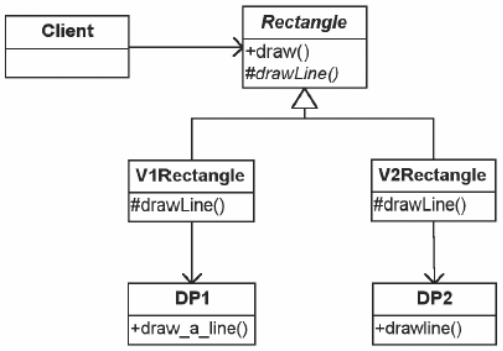
- آنچه را که تغییر می‌کند را پیدا کن و آن را لفاف‌بندی[3] کن.
- ترکیب[4] اشیاء را بر ارث‌بری ترجیح بده.

مساله:

فرض کنید به ما گفته شده‌است که برنامه‌ای بنویسید که بتواند با استفاده از دو برنامه‌ی نقاشی کار کند و مستطیل بکشد و در زمان درست کردن مستطیل است که می‌فهمیم از کدام برنامه باید استفاده کنیم(DP1 یا DP2).

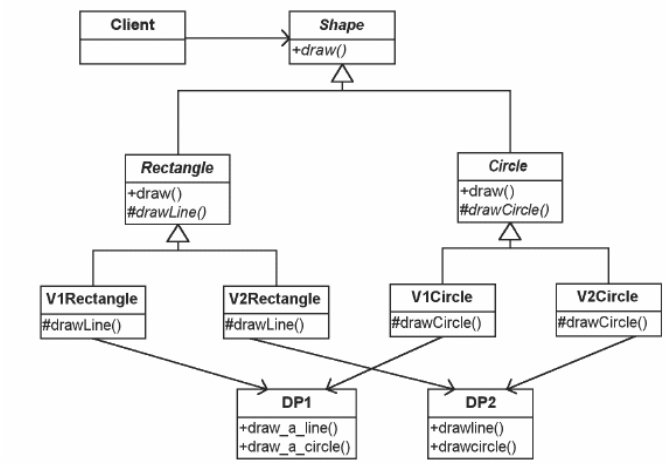
	DP1	DP2
Used to draw a line	`draw_a_line( x1, y1, x2, y2)`	`drawline( x1, x2, y1, y2)`
Used to draw a circle	`draw_a_circle( x, y, r)`	`drawcircle( x, y, r)`

می‌توان دو نوع شیء مستطیل داشت که یکی از DP1 و دیگری از DP2 استفاده می‌کنند. هردوی آن‌ها متد draw را خواهند داشت ولی آن را متفاوت پیاده‌سازی خواهند کرد.



[1] Abstraction
[2] Implementation
[3] Encapsulate
[4] Composition

حالا فرض کنید قرار است نوع دیگری از شکل‌ها مثلا دایره را هم پشتیبانی کنیم و استفاده کننده نباید بتواند فرق مستطیل و دایره را تشخیص دهد. طبق روال قبل خواهیم داشت:



مشکل:

این راه حل از تعدد ترکیب‌ها[5] رنج می‌برد.

حال فرض کنیم یک برنامه‌ی نقاشی جدید(یک تفاوت دیگر در پیاده‌سازی) هم داشته باشیم. در نتیجه شش نوع شکل(حاصل ضرب دو نوع شکل و سه نوع برنامه‌ی نقاشی) خواهیم داشت. اگر یک شکل دیگر(یک تفاوت مفهومی دیگر) هم اضافه شود نه نوع شکل و ...

مشکل تعدد کلاس‌ها به این دلیل اتفاق می‌افتد که تجرید(نوع شکل) و پیاده‌سازی(برنامه‌ی نقاشی) شدیدا به هم چفت شده‌اند[6]. هر نوع از شکل‌ها باید نوع برنامه‌ی نقاشی‌ای که استفاده می‌کند را بداند. برای اینکه تعداد کلاس‌ها به صورت خطی تغییر کنند باید تغییر در پیاده‌سازی و تجرید را از هم جدا کرد.

مشکلات طراحی:

- آیا افزونگی[7] دیده می‌شود؟
- پیوستگی[8] بالاست یا پایین؟
- اشیاء شدیدا به هم چفت شده‌اند یا نه؟

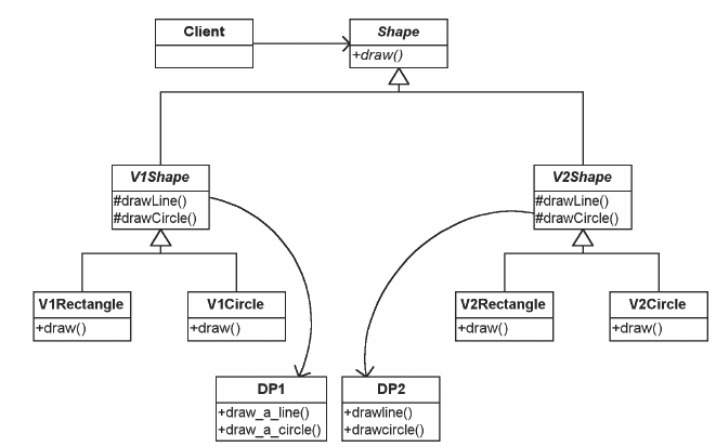
ارث‌بری خوب است ولی خیلی هم نباید از آن استفاده کرد.

“Given this new hammer(Inheritance), everything seems like a nail”

---

[5] Combinatorial Explosion
[6] Tightly coupled
[7] Redundancy
[8] Cohesion

راه حل دیگری که ممکن است به نظر برسد به این ترتیب است:

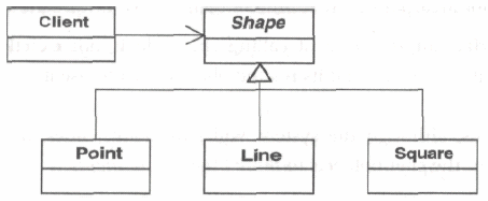


گرچه از میزان چفت‌شدگی کاسته شده‌است ولی هنوز مشکل تعدد کلاس‌ها پابرجاست. با مقدم کردن ارث‌بری برای برنامه‌های نقاشی مختلف افزونگی برای ارتباط به DP1 و DP2 از بین رفت ولی افزونگی وجود دو نوع از مستطیل‌ها و دو نوع از دایره‌ها برجا مانده‌است. دقت کنید که هر دو جفت آن‌ها متد draw مشابهی دارند.

الگوی پل وقتی سودمند است که ما تجریدی داشته‌باشیم که پیاده‌سازی‌های مختلفی داشته باشد. این الگو اجازه می‌دهد که تجریدها و پیاده‌سازی‌ها مستقل از هم تغییر کنند.

مروری بر لفاف‌بندی:

اکثر ما لفاف‌بندی را به عنوان پنهان کردن اطلاعات می‌آموزیم. متاسفانه این تعریف بسیار محدودکننده است. درست است که لفاف‌بندی می‌تواند اطلاعات را پنهان کند، اما علاوه بر آن به صورت‌های دیگری هم می‌تواند استفاده شود.



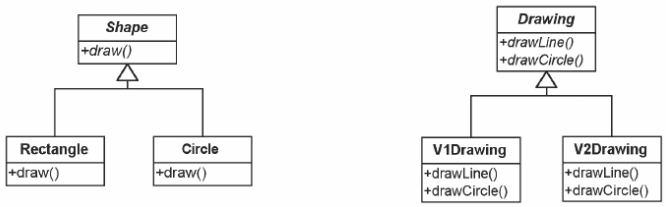
در این شکل می‌توان دید که لفاف‌بندی در سطوح متفاوتی استفاده می‌شود. درست است که از آن برای پنهان‌سازی اطلاعات اشیاء مختلف می‌توان استفاده کرد اما باید به این هم دقت کرد که استفاده‌کننده از نوع شیء خاصی که استفاده می‌کند با

خبر نیست. بنابراین می‌بینیم که کلاس‌های واقعی به گونه‌ای پنهان شده و در لفاف قرار گرفته‌اند. عملا ما چیزی را که تغییر می‌کرده را پیدا کرده‌ایم و آن را در پشت یک کلاس مجرد پنهان کرده‌ایم(قانون کلی اول).

استخراج الگوی پل:

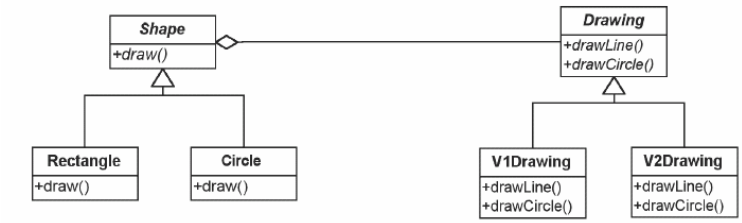
ابتدا چیزهایی را که تغییر می‌کنند را پیدا می‌کنیم. در این مورد انواع شکل‌های مختلف و برنامه‌های نقاشی مختلف را داریم.

شکل‌ها مسئولیت دارند که بدانند چگونه خودشان را رسم کنند. در سوی دیگر برنامه‌های نقاشی هستند که مسئولیت کشیدن خط و دایره را بر عهده دارند. در ادامه تفاوت‌های موجود را وارد می‌کنیم:

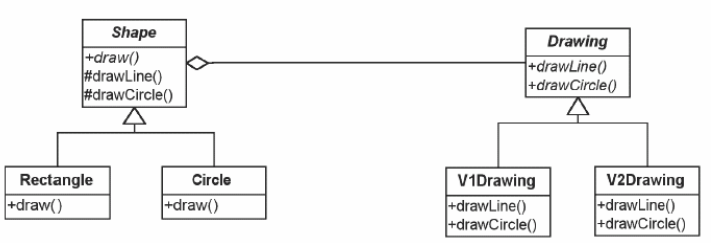


V1Drawing از DP1 و V2Drawing از DP2 استفاده می‌کند. برای سادگی فعلا آن‌ها را نمایش نمی‌دهیم.

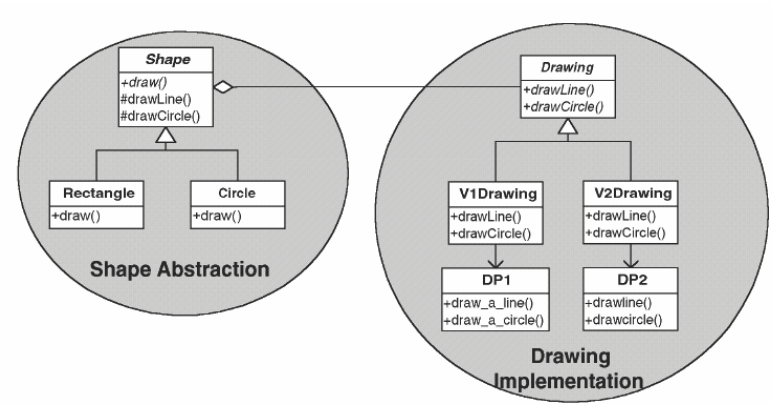
حال می‌خواهیم براساس قانون کلی دوم(ترجیح استفاده از ترکیب) این دو کلاس را به هم ربط دهیم. به هر حال یا شکل‌ها باید از برنامه‌ها استفاده کنند یا برعکس.
ابتدا فرض می‌کنیم برنامه‌ها از شکل‌ها استفاده کنند. اگر برنامه‌های نقاشی بخواهند مستقیما اشیاء را رسم کنند باید کلیاتی از آن‌ها را بدانند. مثلا بدانند آن‌ها چی هستند، یا چه شمایلی دارند که این یکی از اساسی‌ترین قوانین شیءها را نقض می‌کند و آن این که: هر شیء باید تنها مسئول خودش باشد. این کار لفاف‌بندی را نیز نقض می‌کند. برنامه‌ی نقاشی نیاز خواهد داشت اطلاعات خاصی(نوع شکل) را داشته باشد تا آن را رسم کند. دیگر هر شیء فقط مسئول خودش نیست.
حال اگر برعکس عمل کنیم؛ شکل‌ها خودشان را رسم می‌کنند. شکل‌ها نیازی ندارند که نوع برنامه‌ی نقاشی را بدانند. به‌علاوه شکل‌ها مسئولیت کنترل کلاس نقاشی را بر عهده دارند.



در ادامه دو متد drawLine و drawCircle را به کلاس شکل اضافه می‌کنیم که متدهای مشابه از کلاس نقاشی را صدا می‌کند.



این کار برای رعایت قانون یک قانون یک جا[9] انجام شد. این قانون می‌گوید اگر احتمال می‌دهی چیزی در آینده تغییر کند، تمام قوانین مربوط به آن را در یک جا قرار بده چون اگر در جاهای مختلف اعمال شود امکان فراموش کردن تغییر در همه‌ی محل‌ها وجود دارد. درست است که مستطیل یا دایره می‌توانند در متد draw خود مستقیما متد مربوطه از نقاشی مربوط به شکل را صدا بزنند. ولی طبق این قانون بهتر است یک متد برای وکالت[10] در شکل قرار دهیم تا متد مربوطه از نقاشی را صدا بزند تا همه‌ی کارها در همان جا انجام شوند.



نتیجه کاملا شبیه حل مساله با ارث‌بری است فقط متدها هم‌اکنون در شیءهای مختلف پخش شده‌اند. الگوی پل ما را قادر می‌سازد تا پیاده‌سازی را چیزی خارج از شیء خود ببینیم، چیزی که توسط شیء ما استفاده می‌شود[11]. مانند این دید را در قانون وابستگی معکوس نیز داشتیم. در شکل عملا سمت چپ شامل تغیر در مفاهیم مجرد[12] است و سمت راست دگرگونی در روشی که مفاهیم مجرد پیاده‌سازی می‌شوند را در بر می‌گیرد. عملا تغییرات در شکل‌ها توسط کلاس شکل لفاف‌بندی شده است و تغییرات در برنامه‌ی نقاشی توسط کلاس نقاشی.

توجه داشته باشید که در شکل بالا از الگوی انطباق‌دهنده[13] استفاده شده است؛ این اتفاق به این دلیل افتاده است که برنامه‌های نقاشی به ما داده شده‌اند و ما باید از آن‌ها استفاده کنیم. این برنامه‌ها از قبل واسط دارند پس از الگوی انطباق‌دهنده برای انطباق دادن آن‌ها استفاده می‌کنیم تا بتوانیم به یک صورت با آن‌ها برخورد کنیم.

---

[9] One Rule One Place
[10] Delegation
[11] Used by
[12] Abstractions
[13] Adapter Pattern