بسمه تعالی

درس مهندسی نرم‌افزار پیشرفته

فصل سیزدهم

الگوهای طراحی

دکتر فریدون شمس

اهداف جلسه

- بررسی مفاهیم الگوهای طراحی
- شناخت کاربردهای الگوهای طراحی
- آشنایی با چند الگوی طراحی
- آشنایی با ضدالگوها (*AntiPattern*)
- معرفی ضدالگوهای متداول

فهرست مطالب

- عوامل ایجاد الگوی طراحی
- ساختارهای طراحی تکرارشونده
- مفهوم الگوی طراحی
- تاریخچه الگوهای طراحی
- ساختار الگوی طراحی
- معرفی چند الگو طراحی
- ضدالگوها

عوامل ایجاد الگوهای طراحی

- روش‌های تحلیل و طراحی شی‌گرا تاکید بسیاری بر استفاده از نمادهـا در طراحی دارند
  - برای مستندسازی و ذکر خصوصیات مناسب هستند
- اما تحلیل و طراحی شی‌گرا تنها رسم نمودار نیست
  - نقاشی خوب دلیل بر طراحی خوبی نیست!
- طراحی شی‌گرای خوب نیاز به سال‌ها تجربه دارد
  - طراحی به اندازه دانستن گرامر زبان اهمیت دارد
- بیشترین استفاده مجدد در هنگام طراحی اتفاق می‌افتد
  - مشکلات با تجربه طراحی رفع می‌شوند

ساختارهای طراحی تکرار شونده

- ساختارهای تکرارشونده سیستم‌های شی‌گرا سبب ارتقای
  - تجرید (*Abstraction*)
  - انعطاف‌پذیری (*Flexibility*)
  - واحدبندی (*Modularity*)
  - ظرافت (*Elegance*)
- که حاوی اطلاعات ارزشمند طراحی هستند

اما نمی‌توان از این ساختارها به صورت مستقیم استفاده نمود

مسئله اصلی: شنـــاسایی، دسته‌بندی و استفـــاده از آنهاست

الگوی طراحی

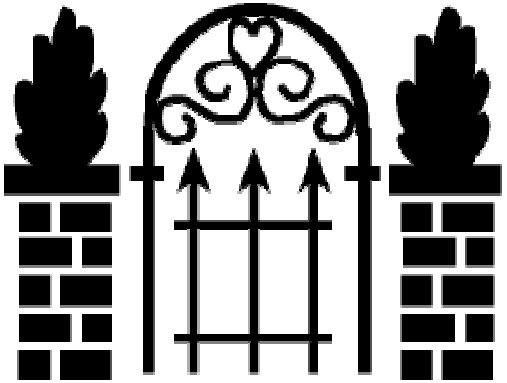
- تجریدی از ساختار طراحی تکرارشونده است
- شامل کلاس و/یا اشیاء
  - وابستگی‌ها (*Dependencies*)
  - ساختارها (*Structures*)
  - تعاملات (*Interactions*)
  - قراردادها (*Conventions*)
- نام‌ها و ساختار طراحی را به‌صورت صریح تعیین می‌کند
- چکیده‌ای از تجربیات طراحی است

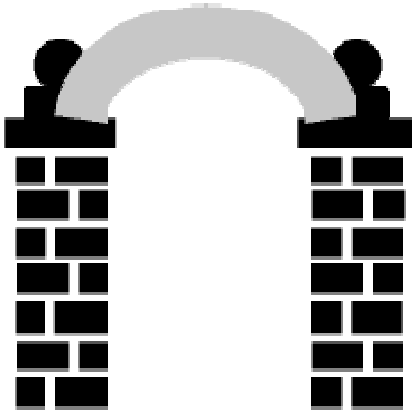
الگوی طراحی (ادامه)

- مجموعه‌ای از بهترین تجربیات است
- راه‌حلی نمونه برای مسئله‌ای در زمینه خاص است
- راهی برای تسهیل ارتباطات بین ذینفعان
- الگوها پیدا شده‌اند و ابداع نشده‌اند

تاریخچه الگوهای طراحی

- *Christopher Alexander* –۱۹۶۴
  - کتاب معماری و طراحی شهری
  - بیان نکاتی در مورد ترکیب شکل‌ها
  - زبان الگو
  - راهی برای ساختن سریع



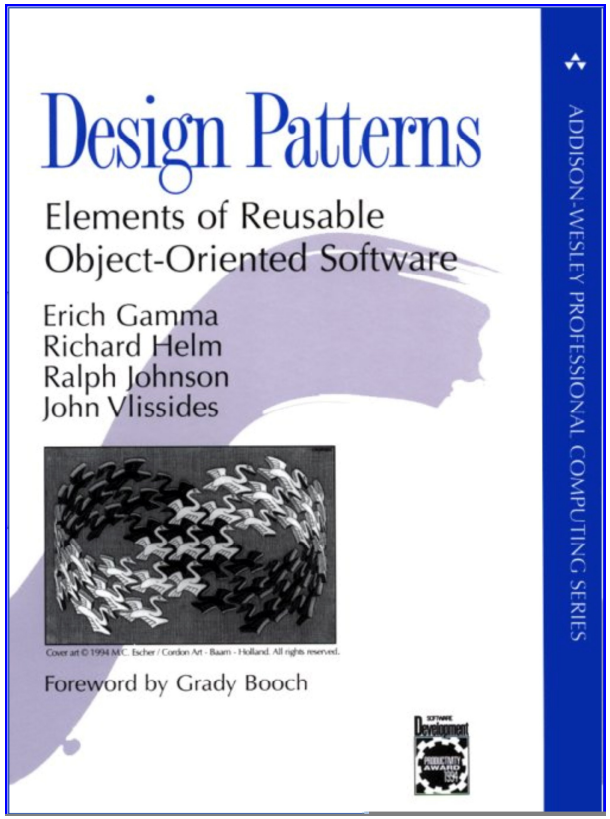


تاریخچه الگوهای طراحی (ادامه)

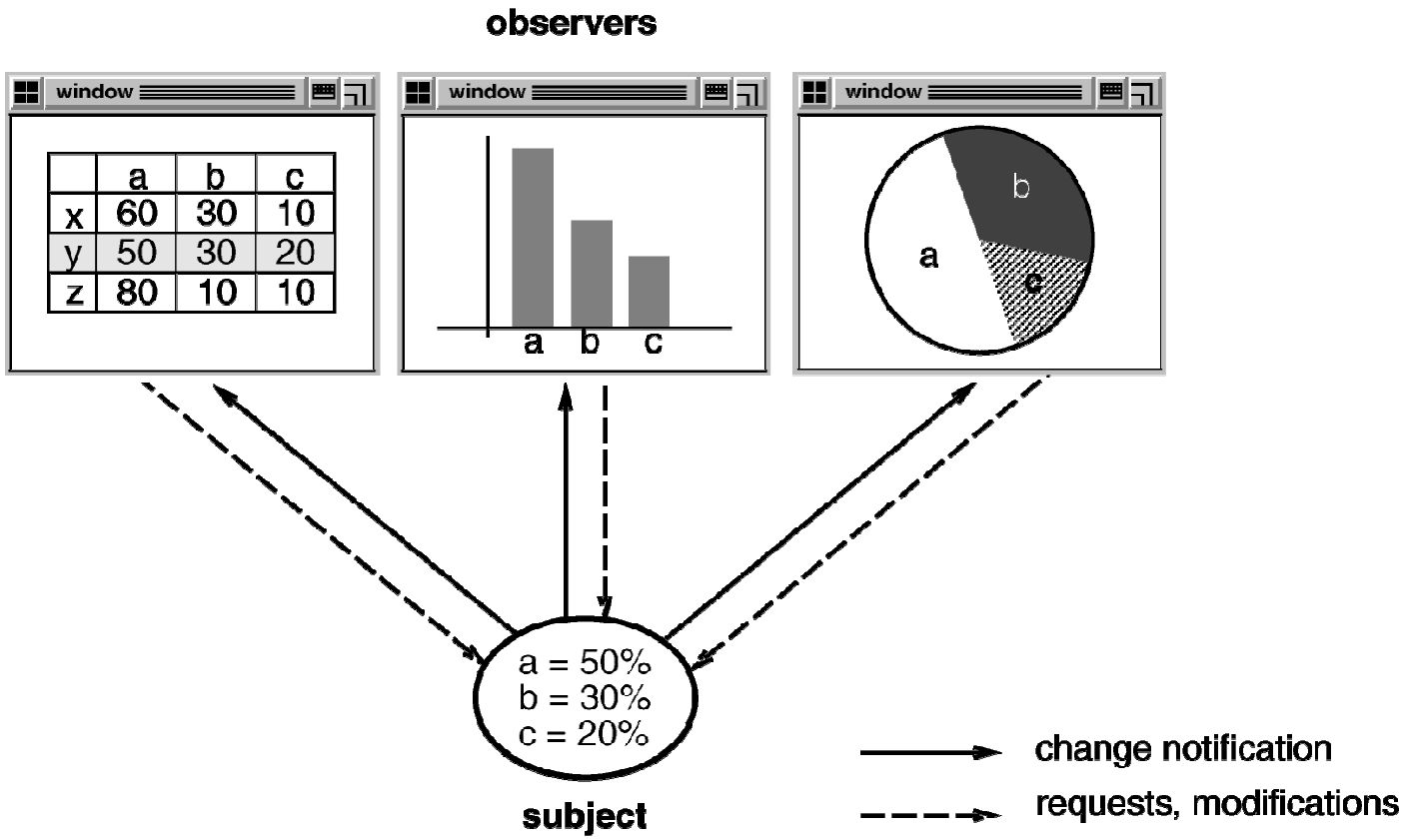
- ***Ward Cunningham & Kent Beck*** –۱۹۸۷
  - تصمیم به استفاده از ایده‌های الکساندر
  - توسعه پنج الگو برای برنامه‌نویسان مبتدی ***Smalltalk***
  - ارائه مقاله‌ای در این زمینه در کنفرانس ***OOPSLA'87***

تاریخچه الگوهای طراحی (ادامه)

- ۱۹۹۵– *Erich Gamma*، *Richard Helm*، *Ralph Johnson* و *John Vlissides*
  - طراحی نرم‌افزار
  - ارائه کتاب الگوهای طراحی



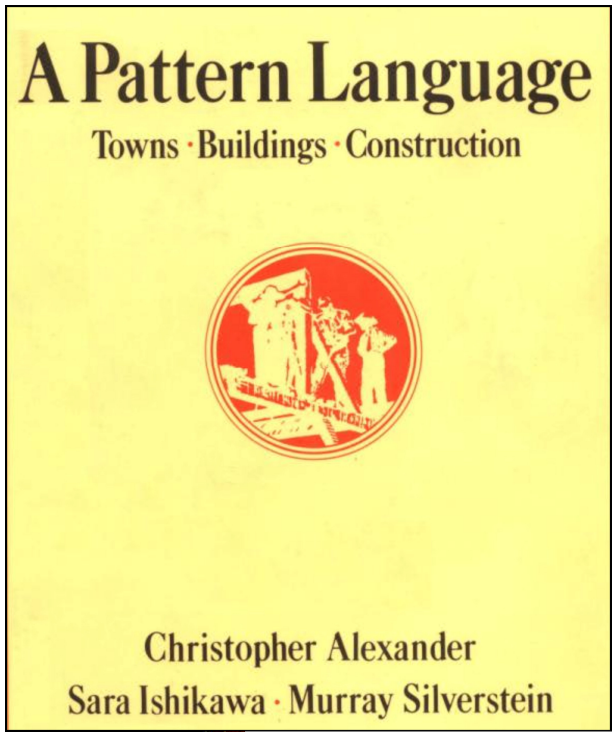
نمونه الگوی طراحی



ساختار الگوی طراحی

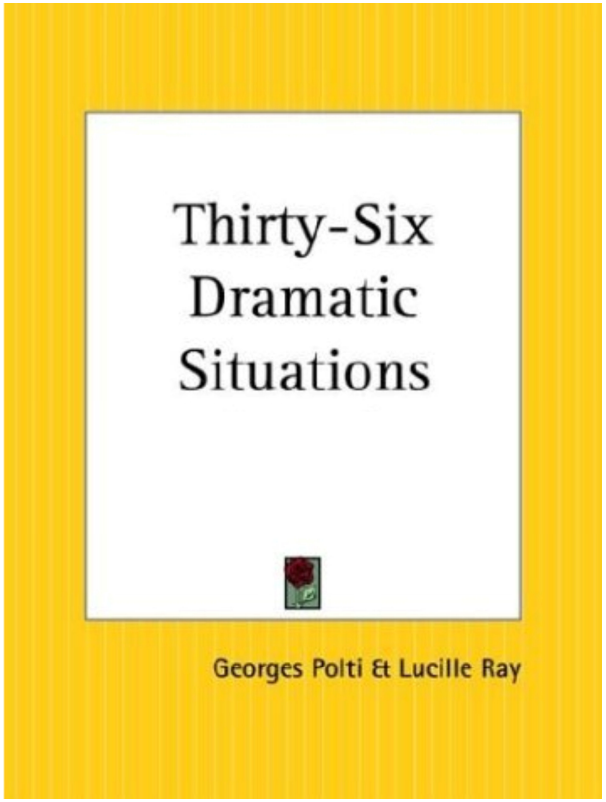
- ساختارهای متعددی برای بیان الگوهای طراحی پیشنهاد شده است
- چهار بخش اصلی هر الگوی طراحی
  - نام الگو
  - مسئله (شامل اجبارهای مسئله)
  - راه‌حل
  - نتایج و کاربردها

ساختار الگوی طراحی (ادامه)



- ساختار پیشنهادی *Alexander*
  - نام
  - زمینه
  - مسئله (اجبارها)
  - راه‌حل (پیکربندی و چگونگی راه‌حل)
  - بیان ارتباط بین زمینه، مسئله و راه‌حل

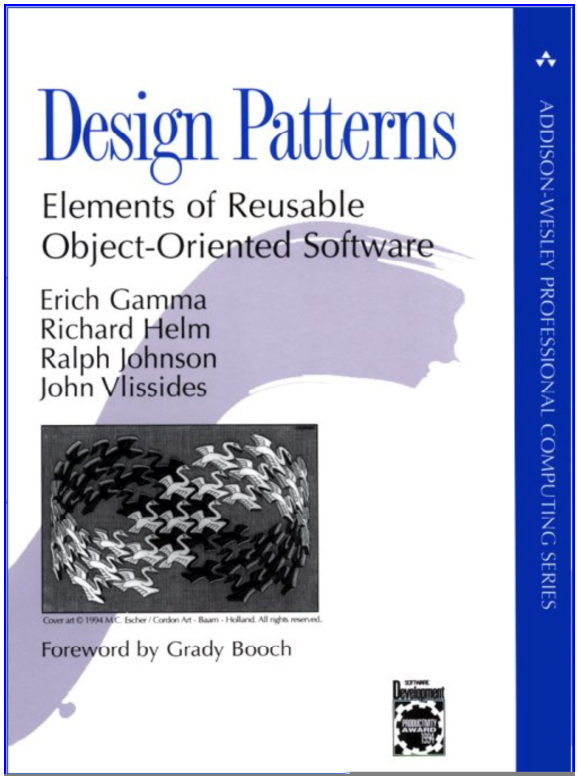
ساختار الگوی طراحی (ادامه)



- ساختار پیشنهادی *Polti*
  - نام
  - عناصر
  - توصیف
  - نمونه‌ها
  - موارد استفاده
  - جزئیات

ساختار الگوی طراحی (ادامه)

- ساختار پیشنهادی *GoF*
  - نام
  - طبقه‌بندی
  - منظور
  - اسامی مشابه
  - انگیزه
  - کاربرد
  - ساختار
  - شرکاء
  - همکاری
  - نتایج
  - پیاده‌سازی/نمونه کد
  - موارد استفاده
  - الگوهای مرتبط



ساختار الگوی طراحی (ادامه)

- نام (*Name*)
  - بطور خلاصه و موجز کاربرد الگو را نشان می‌دهد
- طبقه‌بندی (*Classification*)
  - طبقه‌بندی براساس کاری که الگو انجام می‌دهد
    - *Creational*: بر روی فرآیند ایجاد اشیا و کلاس‌ها تمرکز دارند
    - *Structural*: بر روی ترکیب کلاس‌ها و اشیا تمرکز دارند
    - *Behavioral*: بر روی رفتار کلاس‌ها و اشیا و توزیع وظیفه تمرکز دارند
  - حوزه کاربرد الگوها شامل اشیاء و کلاس‌ها هستند

ساختار الگوی طراحی (ادامه)

Patterns Classification		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

ساختار الگوی طراحی (ادامه)

- منظور
  - جمله کوتاهی که جواب به سوالات زیر است
    - الگو چه کاری انجام می‌دهد؟
    - علت وجودی الگو چیست؟
    - چه مشکل یا مسئله خاصی را حل می‌کند؟
- نام‌های مشابه
  - در صورتی‌که الگو دارای اسامی شناخته شده دیگری است، قید می‌شود

ساختار الگوی طراحی (ادامه)

- انگیزه (*Motivation*)
  - سناریویی که مشکل طراحی و اینکه چگونه کلاس‌ها و اشیای موجود در الگو مشکل را حل می‌کنند را نشان می‌دهد
  - سناریو کمک می‌کند تا درک بهتری از توصیفاتی مربوط به الگو داشته باشید
- کاربردپذیری (*Applicability*)
  - وضعیتی که الگو می‌تواند به کار برده شود و نحوهٔ شناسایی این وضعیت

ساختار الگوی طراحی (ادامه)

- ساختار (*Structure*)
  - نمایشی گرافیکی از کلاس‌ها در الگو با استفاده از نمادگذاری OMT (*Object Modeling Technique*)
  - برای نمایش تعاملات می‌توانید از نمودار ترتیبی یا نمودار همکاری استفاده نمائید
- شرکاء (*Participants*)
  - کلاس‌ها و/یا اشیایی که در الگوی طراحی و انجام وظایفش مشارکت می‌کنند

ساختار الگوی طراحی (ادامه)

- همکاری (*Collaborations*)
  - نحوهٔ همکاری شرکاء در انجام وظایف الگوی طراحی
- نتایج (*Consequences*)
  - در این بخش به سوالات ذیل پاسخ گفته می‌شود
    - چگونه الگو، اهداف مورد نظر خود را برآورده می‌سازد؟
    - چه توازن‌ها و نتایجی با به کار بردن الگو بدست می‌آید؟
    - چه جنبه‌هایی از ساختار سیستم می‌توانند متفاوت باشند؟

ساختار الگوی طراحی (ادامه)

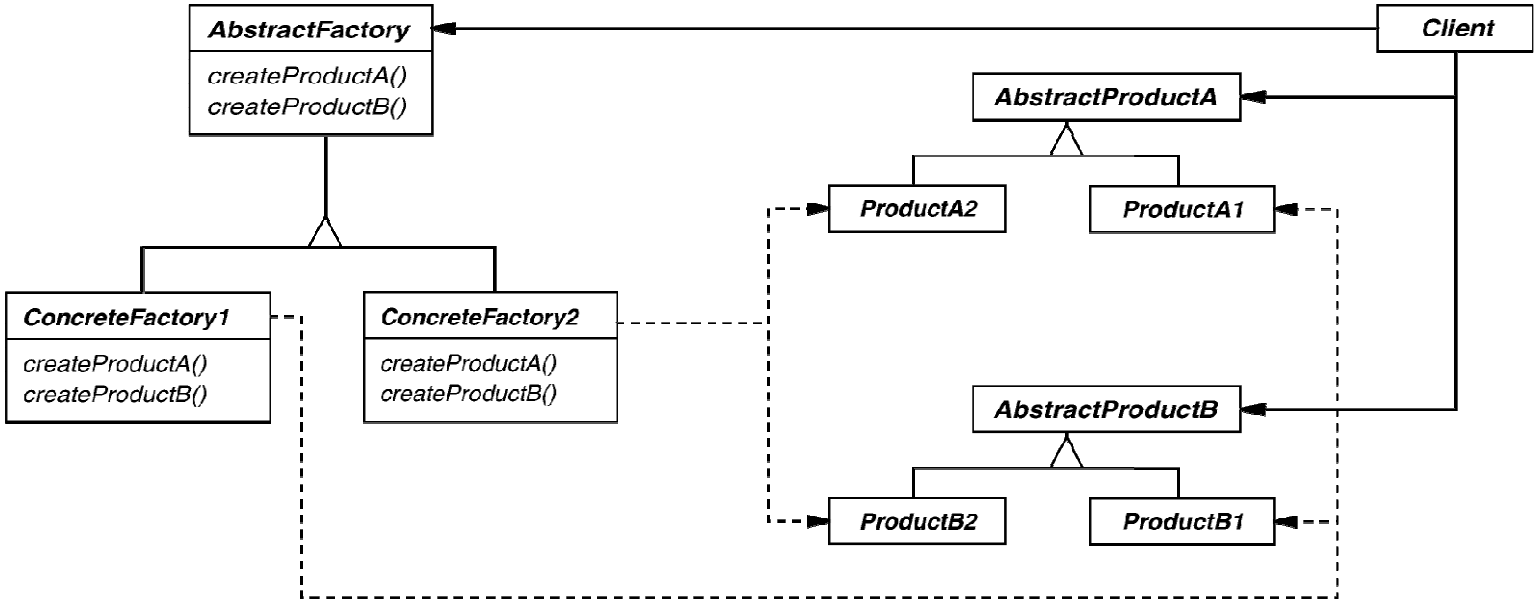
- پیاده‌سازی
  - نکات فنی، نقاط ضعف و دیگر نکاتی که باید در استفاده از الگو مد نظر قرار داد
- نمونه کد
  - بخش‌های از کد الگو به زبان C++ یا *Smalltalk*
- استفاده‌های شناخته شده
  - نمونه‌های از الگو در سیستم‌های واقعی
- الگوهای مرتبط
  - الگوهای که ارتباط نزدیکی با الگوی طراحی مورد نظر دارند

الگوهای طراحی – *Abstract Factory*

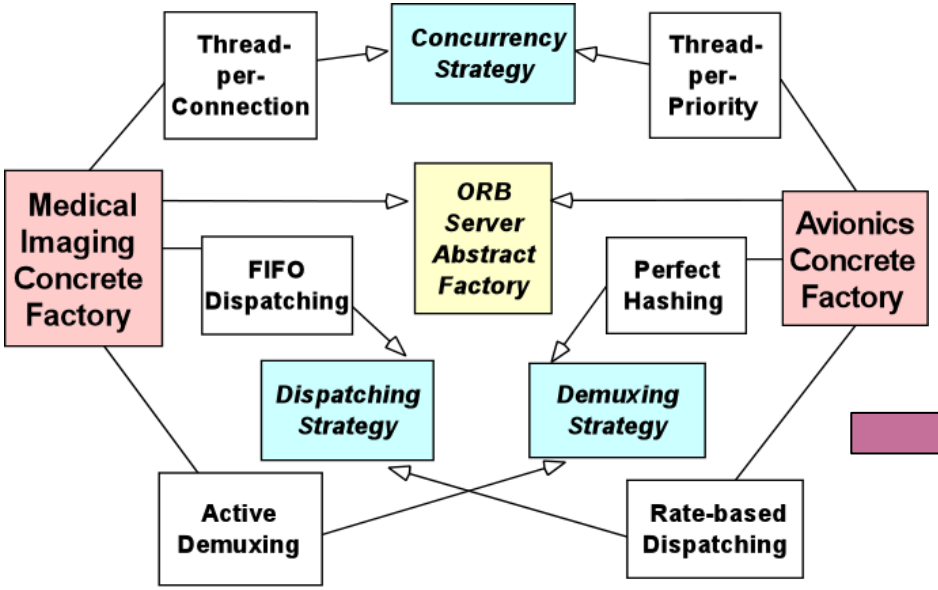
- منظور
  - رابطی برای ایجاد خانواده‌ای از اشیای مرتبط یا وابسته بدون مشخص‌کردن کلاس‌های آنها فراهم می‌کند
- کاربردپذیری
  - وقتی نمی‌توان ساختار ارتباطی بین گروهی از کلاس‌های مرتبط را پیش‌بینی نمود

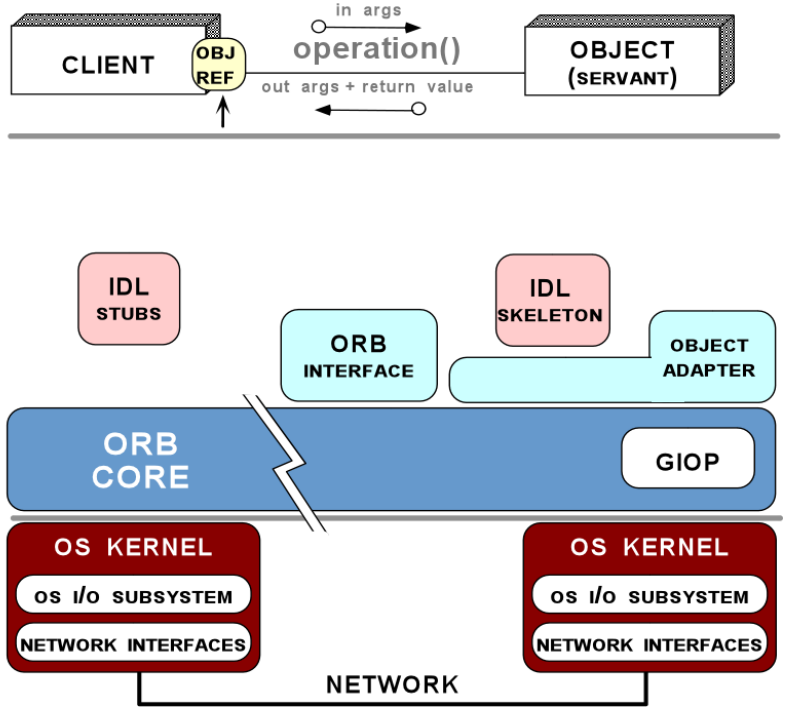
الگوهای طراحی – *Abstract Factory*

- ساختار



Abstract Factory – الگوهای طراحی





الگوهای طراحی – *Abstract Factory*

- نتایج
  - ✓ انعطاف‌پذیری
  - ✓ تجرید
  - ✗ توسعه آن دشوار است
- پیاده‌سازی
  - استفاده از پارامترها راهی برای کنترل اندازه رابط است
  - این الگو در واقع مجموعه‌ای از الگوی *Factory Method* است

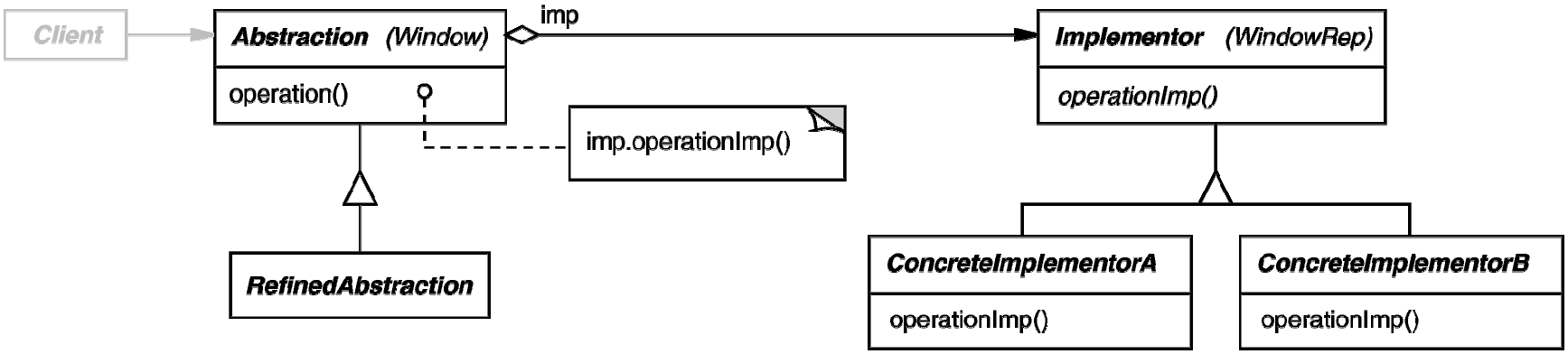
الگوهای طراحی – *Bridge*

- منظور
  - برای جداسازی رابط مجرد(منطقی) از پیاده‌سازی‌اش (فیزیکی) مـورد استفاده قرار می‌گیرد
- کاربردپذیری
  - وقتی رابط و پیاده‌سازی‌اش باید با یکدیگر متفاوت باشند
  - وقتی نیاز به یک رابط واحد برای تعامل با دیگر کلاس‌ها باشد

الگوهای طراحی – *Bridge*

- ساختار

Structure



الگوهای طراحی – *Bridge*

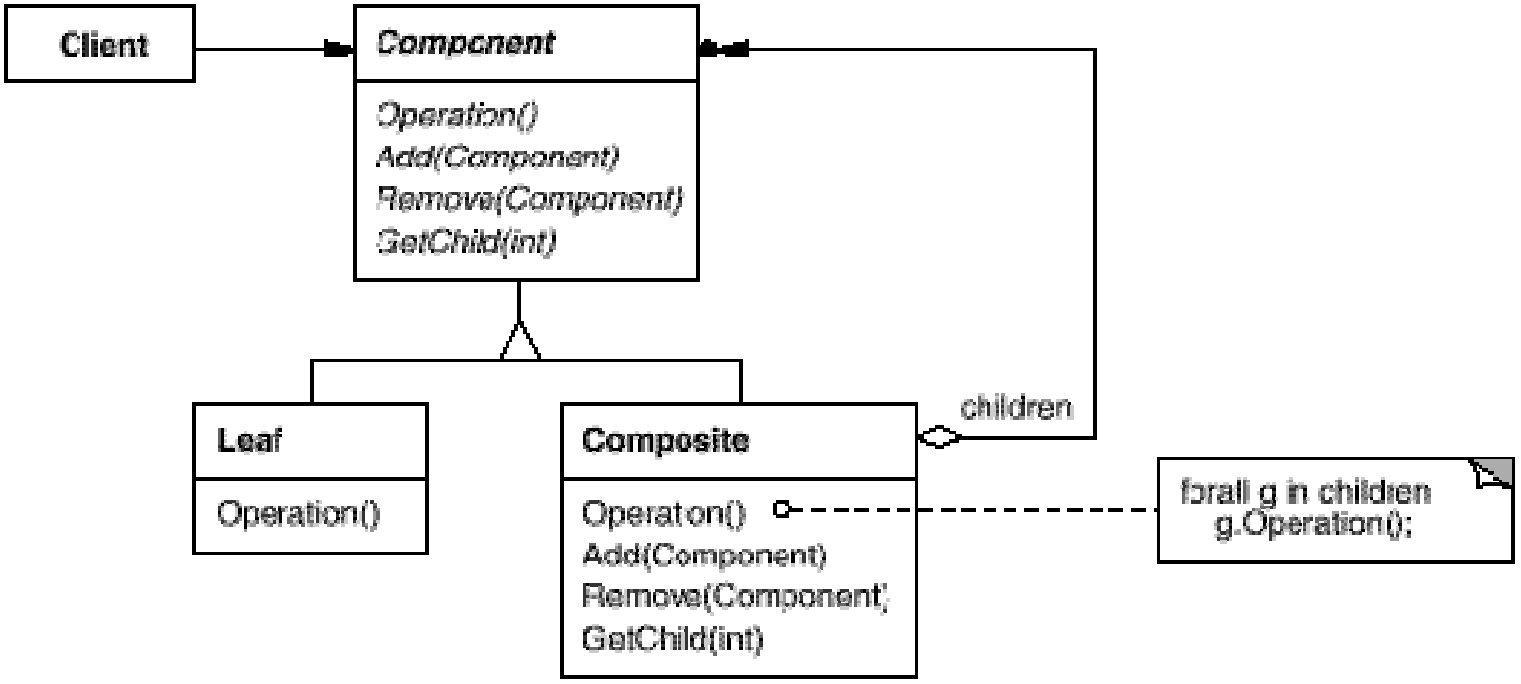
- نتایج
  - ✓ رابط مجرد و پیاده‌سازی از یکدیگر مجزا می‌شوند
  - ✓ پیاده‌سازی می‌تواند مجزا و متغیر باشد
  - ✗ رابط باید به‌گونه‌ای عمومی باشد که همه رابط‌هایی که پیاده‌سازی می‌شوند را حمایت کند
- پیاده‌سازی
  - تنها یک پیاده‌ساز باید وجود داشته باشد
  - ایجاد پیاده‌ساز صحیح دشوار است
  - اشتراک پیاده‌سازها باید مد نظر قرار گیرد

الگوهای طراحی – *Composite*

- منظور
  - ترکیب اشیا در ساختاری درختی برای نمایش سلسله مراتب *Part-Whole*
- کاربردپذیری
  - وقتی اشیا نیاز به ترکیب به‌صورت بازگشتی (*Recursive*) دارند
  - و تفاوتی بین اشیای ترکیبی و ساده وجود ندارند
  - و اشیا در ساختار می‌توانند واحد در نظر گرفته شوند

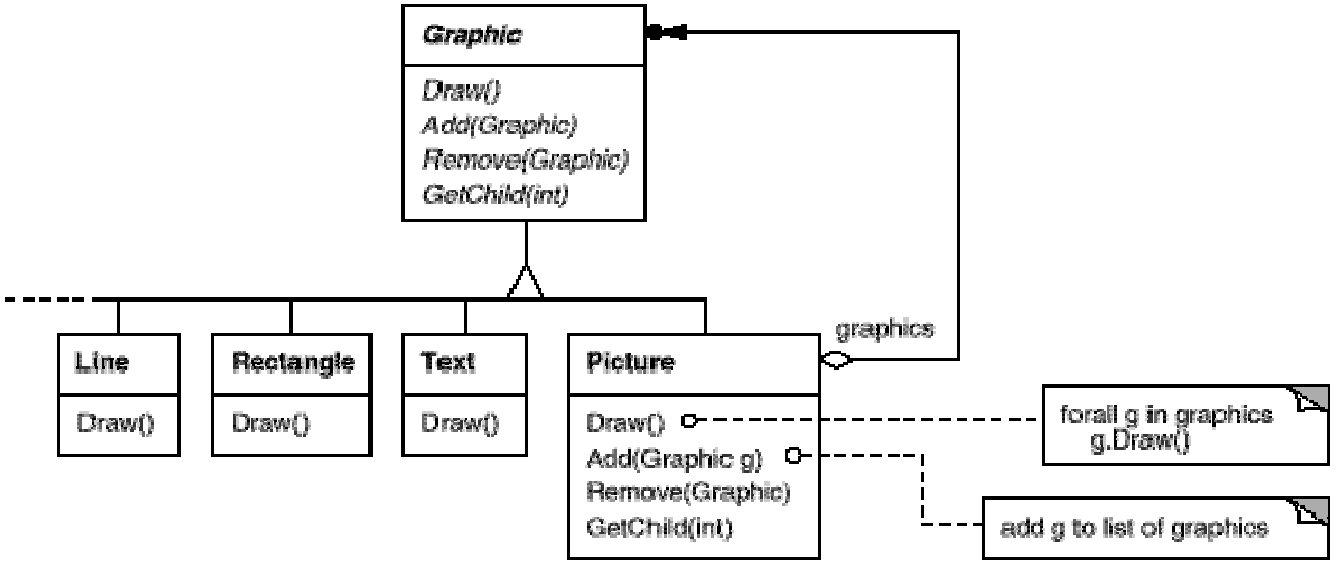
الگوهای طراحی – *Composite*

- ساختار

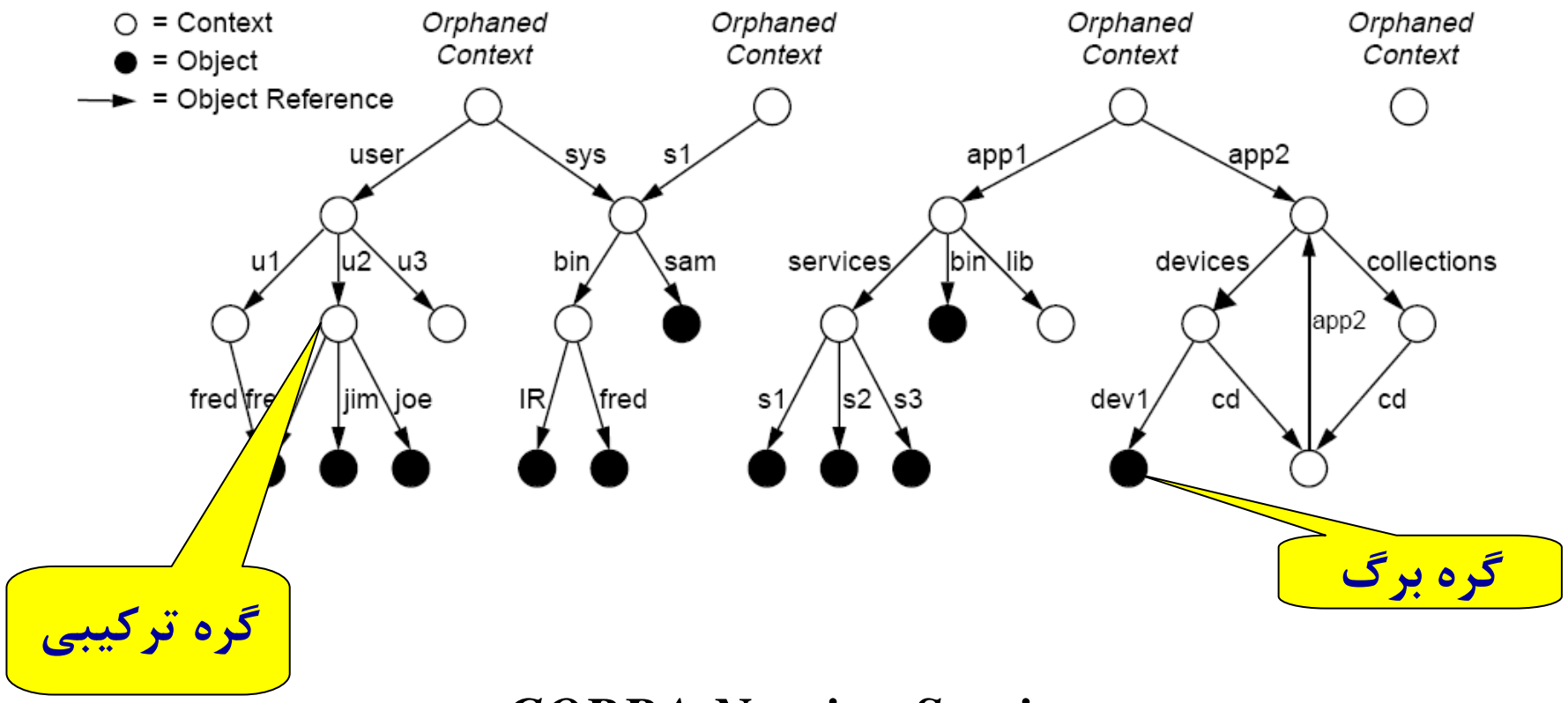


الگوهای طراحی – *Composite*

- کلاس *Graphic* شامل مجموعه‌ای از کلاس‌های دیگر است که تفاوتی با کلاس اصلی ندارند



الگوهای طراحی – *Composite*



CORBA Naming Service

الگوهای طراحی – *Composite*

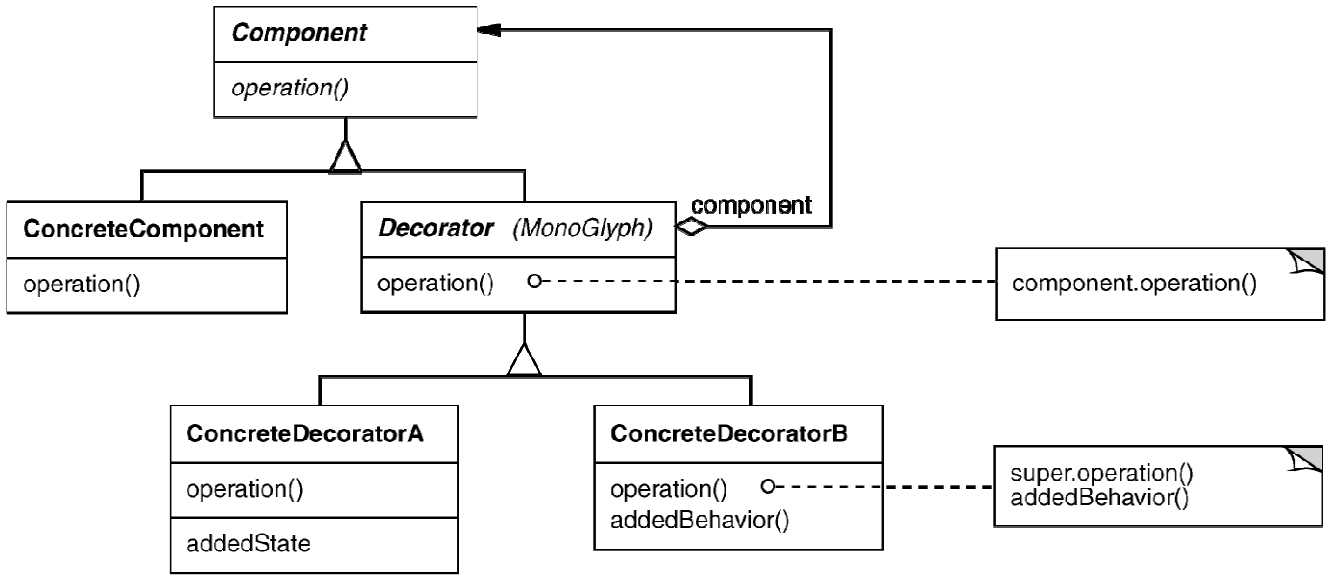
- نتایج
  - ✓ همسانی: بدون در نظر گرفتن پیچیدگی همه اشیا یکسانند
  - ✓ قابلیت توسعه: مولفه‌های جدید براحتی می‌توانند افزوده شوند
  - ✗ سربار: سربار تعداد اشیای جدید را کاهش می‌دهد
- پیاده‌سازی
  - روشی برای شناسایی والدین و فرزندان باید استفاده شود
  - برای والدین و فرزندان از یک رابط استفاده می‌شود
  - حذف یک فرزند مسئولیت دارد! (این فرزند می‌تواند فرزندانی داشته باشد و سبب ایجاد اشیای یتیم (*Orphan*) شود)

الگوهای طراحی – *Decorator*

- منظور
  - افزودن وظیفه جدید به یک شی به صورت پویا
- کاربردپذیری
  - وقتی نیاز به افزودن وظیفه جدید به یک شی به صورت پویا، شفاف و بدون تاثیر بر دیگر اشیا باشد
  - حذف وظیفه خاصی از یک شی به‌صورت پویا
  - وقتی توسعه کلاس به زیرکلاس‌ها غیرممکن باشد

الگوهای طراحی – *Decorator*

- ساختار



الگوهای طراحی – *Decorator*

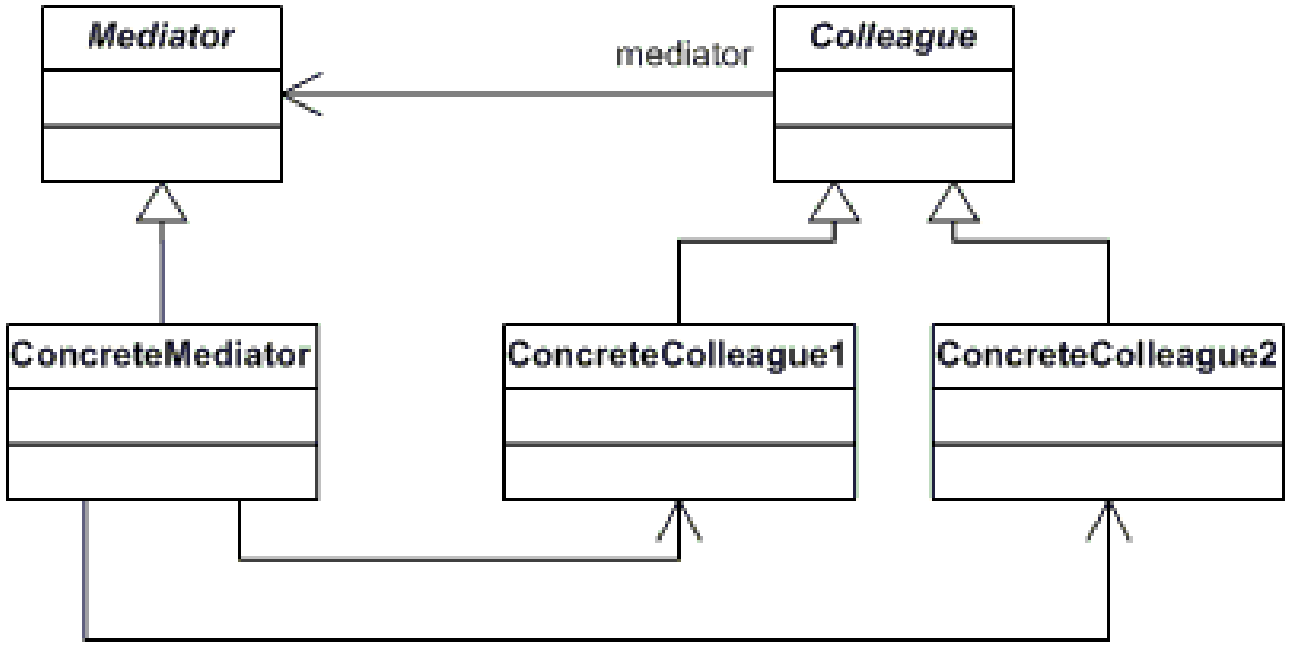
- نتایج
  - ✓ وظایف می‌توانند در زمان اجرا افزوده/حذف شوند
  - ✓ موجب کاهش رشد ساختار سلسله‌مراتبی می‌شود
  - ✗ سبب انسداد رابط می‌شود
  - ✗ ترکیب *decorator* ها بسیار دشوار است
- پیاده‌سازی
  - مطابقت رابط *decorator* ها باید در نظر گرفته شود
  - برای *decorator* از کلاسی سبک و مجرد استفاده نمائید
  - در صورتی‌که نیاز به افزودن یک وظیفه دارید، نیاز به تعریف کلاس برای *decorator* ندارید

الگوهای طراحی – *Mediator*

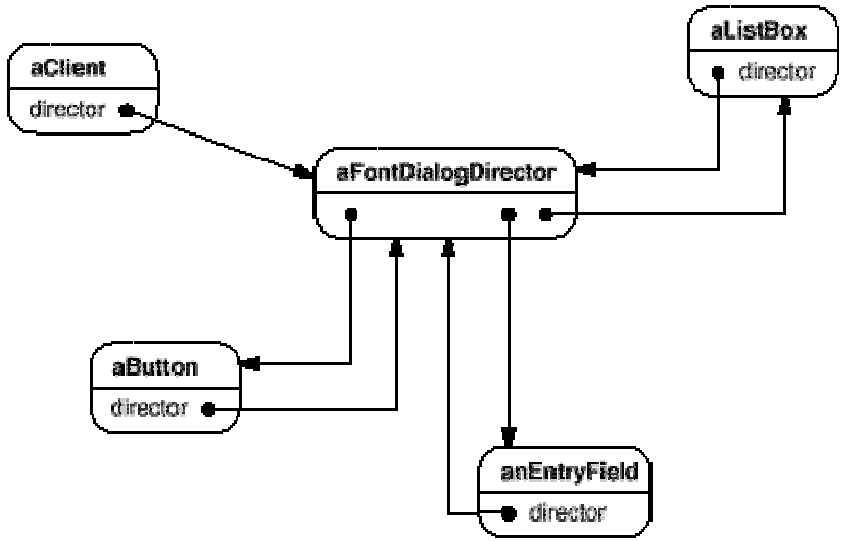
- منظور
  - تعریف شیی که چگونگی تعامل بین اشیاء دیگر را پنهان سازد
- کاربردپذیری
  - وقتی تعدادی شی با یکدیگر به‌صورت خوش‌تعریف تعامل دارند اما تعامل آنها پیچیده است
  - وقتی استفاده مجدد یک شی به‌خاطر تعامل زیاد آن با دیگر اشیا دشوار باشد
  - وقتی رفتاری بین چند کلاس توزیع شده است و باید بدون استفاده از زیرکلاس‌ها سفارشی شود

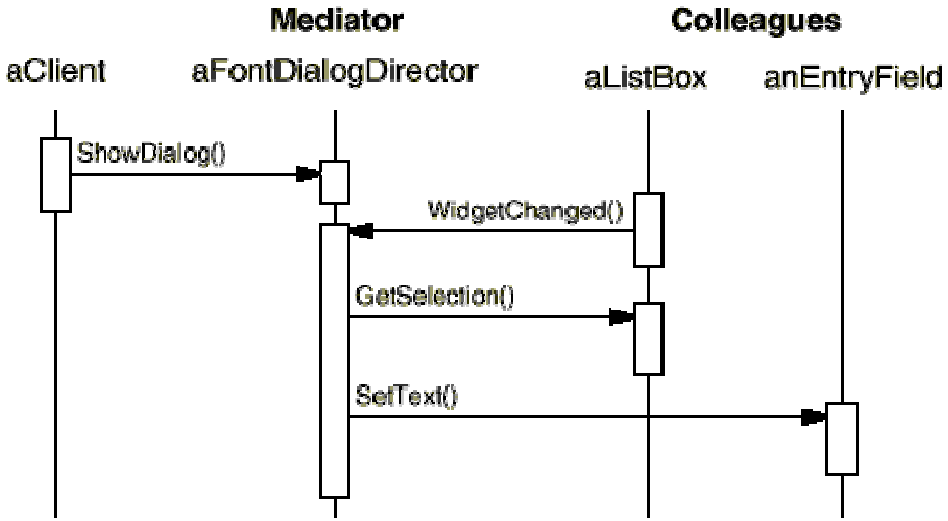
الگوهای طراحی – *Mediator*

- ساختار



الگوهای طراحی – *Mediator*





الگوهای طراحی – *Mediator*

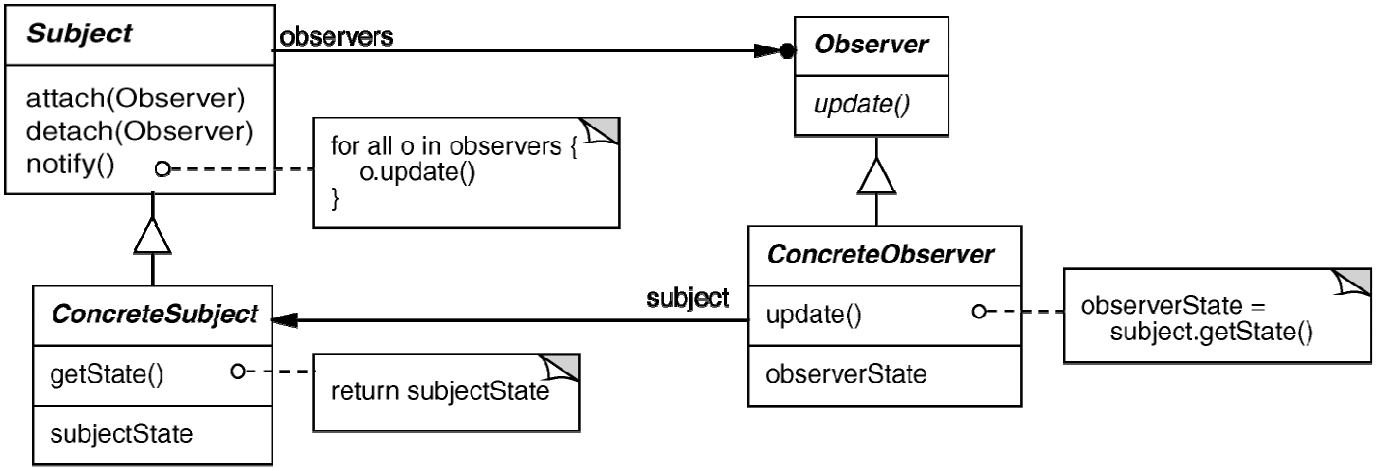
- نتایج
  - ✓ تعداد زیرکلاس‌ها را کاهش می‌دهد
  - ✓ اتصال سست بین کلاس‌های مجاور را افزایش می‌دهد
  - ✓ ارتباط چند به چند را به ارتباط یک به یک تبدیل می‌کند
  - ✗ یک کنترل‌کننده مرکزی ایجاد می‌کند
- پیاده‌سازی
  - نیازی به تعریف کلاس مجرد *Mediator* نیست زیرا کلاس‌های همکار می‌توانند با یک *Mediator* کار کنند

الگوهای طراحی – *Observer*

- منظور
  - تعریف یک رابطه یک به چند بین اشیاء بصورتی‌که وقتی یک شی حالتش را تغییر می‌دهد، بقیه اشیا مطلع شده و بروز شوند
- کاربردپذیری
  - وقتی تجریدی دارای دو جنبه بوده و یکی به دیگری وابسته باشد
  - وقتی تغییر در یک شی سبب تغییر دیگر اشیا می‌شود و نمی‌دانید چه تعداد شی دیگر نیاز به تغییر دارند
  - وقتی یک شی نیاز به مطلع ساختن شی دیگر دارد و در مورد چگونگی آن شی اطلاعی ندارد

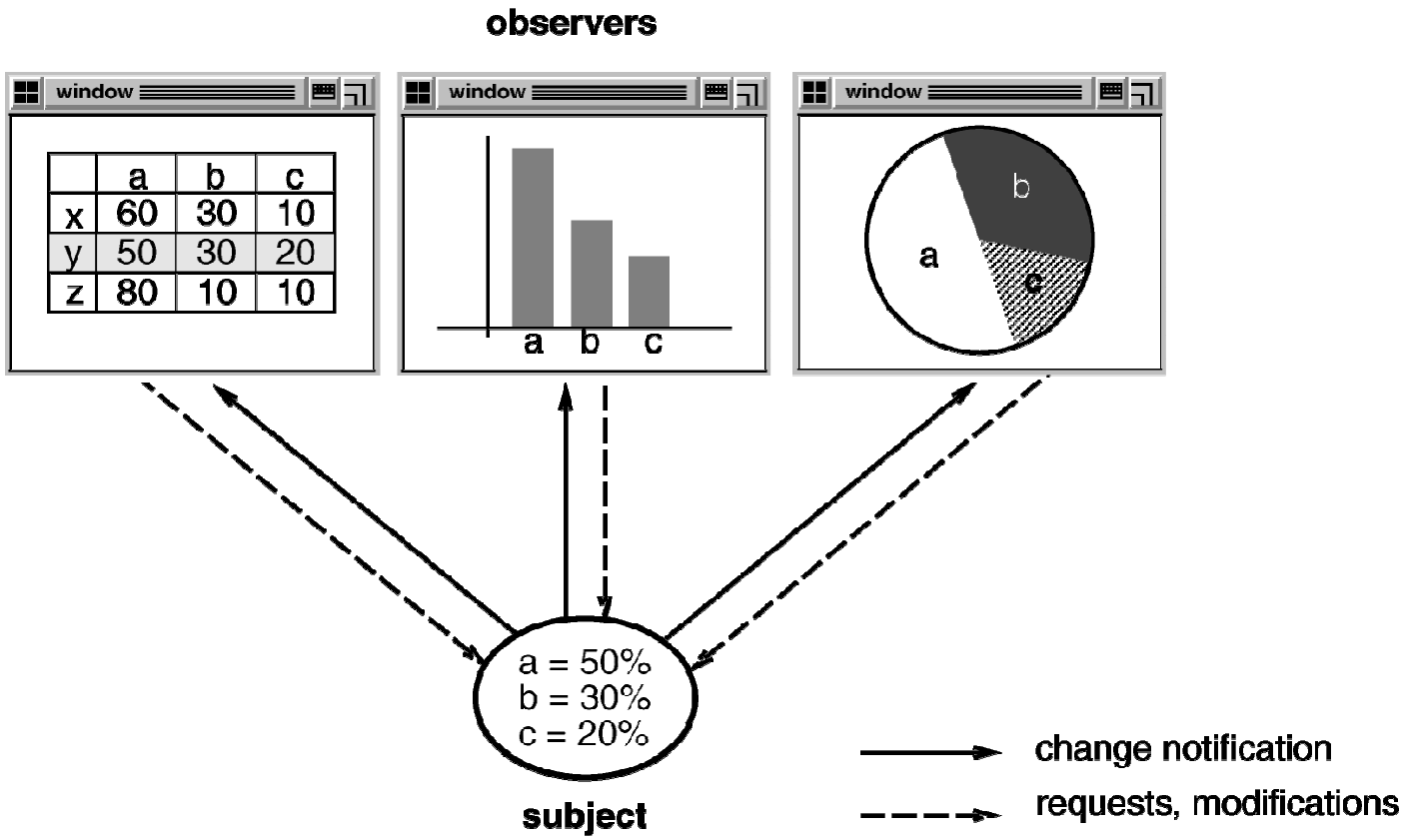
الگوهای طراحی – *Observer*

- ساختار



الگوهای طراحی – *Observer*

- مثال



الگوهای طراحی – *Observer*

- نتایج

✓ واحدبندی: *Subject* و *Observer*ها می‌توانند مستقل باشند

✓ قابلیت توسعه: می‌توان هر تعداد *Observer* تعریف نمود

✓ قابلیت سفارشی شدن: *Observer*هـای متفـاوت، دیـدهای مختلـف را نمایش می‌دهند

✗ بروزرسانی غیرمنتظره: *Observer*ها اطلاعی از یکدیگر ندارند

✗ سربار بروزرسانی: بروزرسانی سربار زیادی ایجاد می‌نماید

الگوهای طراحی – *Observer*

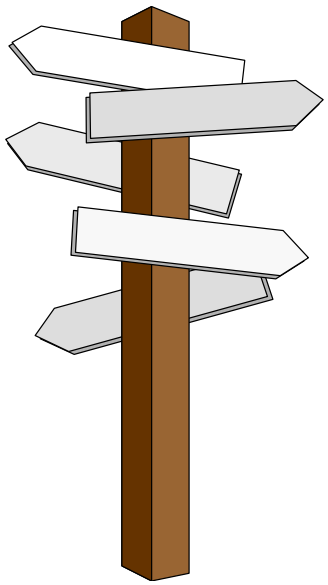
- پیاده‌سازی
  - نگاشت *Subject*ها به *Observer*های متناظرشان راهی ساده برای در جریان بودن *Subject* از بروزرسانی *Observer* است
  - ارجاع معلق به *Subject*های حذف شده باید مورد توجه قرار گیرند
  - از پروتکل‌های بروزرسانی ویژه *Observer* (مدل‌های *push and pull*) اجتناب نمائید

ضدالگوها (*AntiPatterns*)

- ایده ضدالگوها از آنجا ناشی شده است که اغلب کارهای منتشر در مهندسی نرم‌افزار به راه‌حل‌های سازنده و موثر تمرکز دارد
- ضدالگوها بر راه‌حل‌های منفی و ناموفق تمرکز دارند
- ضدالگو مجموعه‌ای از راه‌حل‌های متداول که برای حل یک مسئله اتفاق می‌افتند و بطورقطعی نتایج منفی یا ناموفق تولید نموده‌اند را توصیف می‌نماید

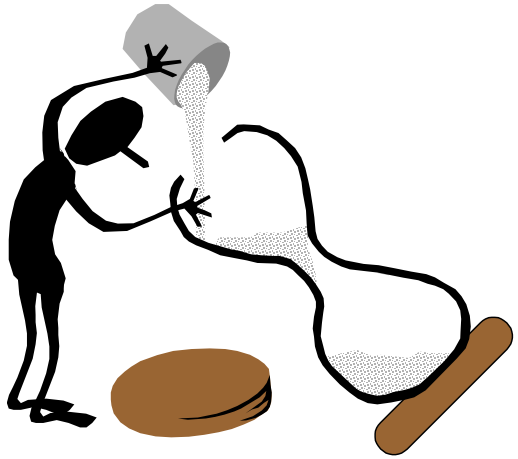
ضدالگوها (ادامه)

- همانطور که نیاز به دانستن الگوها در زمینه خاص وجود دارد، دانستن وجود یا عدم وجود ضدالگو می‌تواند به توسعه بهتر نرم‌افزار کمک کند
  - بسیاری از مهندسان نرم‌افزار می‌خواهند بدانند «آیا راهی که می‌روند به شکست منجر می‌شود»



مثالی از ضدالگوها

- در سطح مدیریت پروژه
  - پنج پروژه از شش پروژه به شکست منجر شده‌اند!
  - یک سوم پروژه‌های نرم‌افزاری رها شده‌اند!
  - بودجه و زمان واقعی انجام پروژه بیش از دو برابر از آنچه تخمین زده می‌شود، است!!!



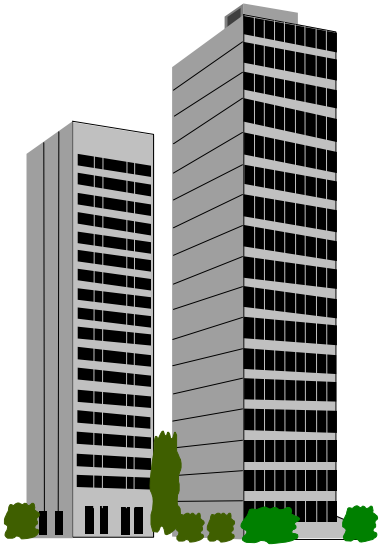
مثالی از ضدالگوها (ادامه)

- کار دیروز
  - برنامه‌نویسی ساختاریافته
  - طراحی بالا به پائین
  - معماری *Client/Server*
  - تولید کد از مدل

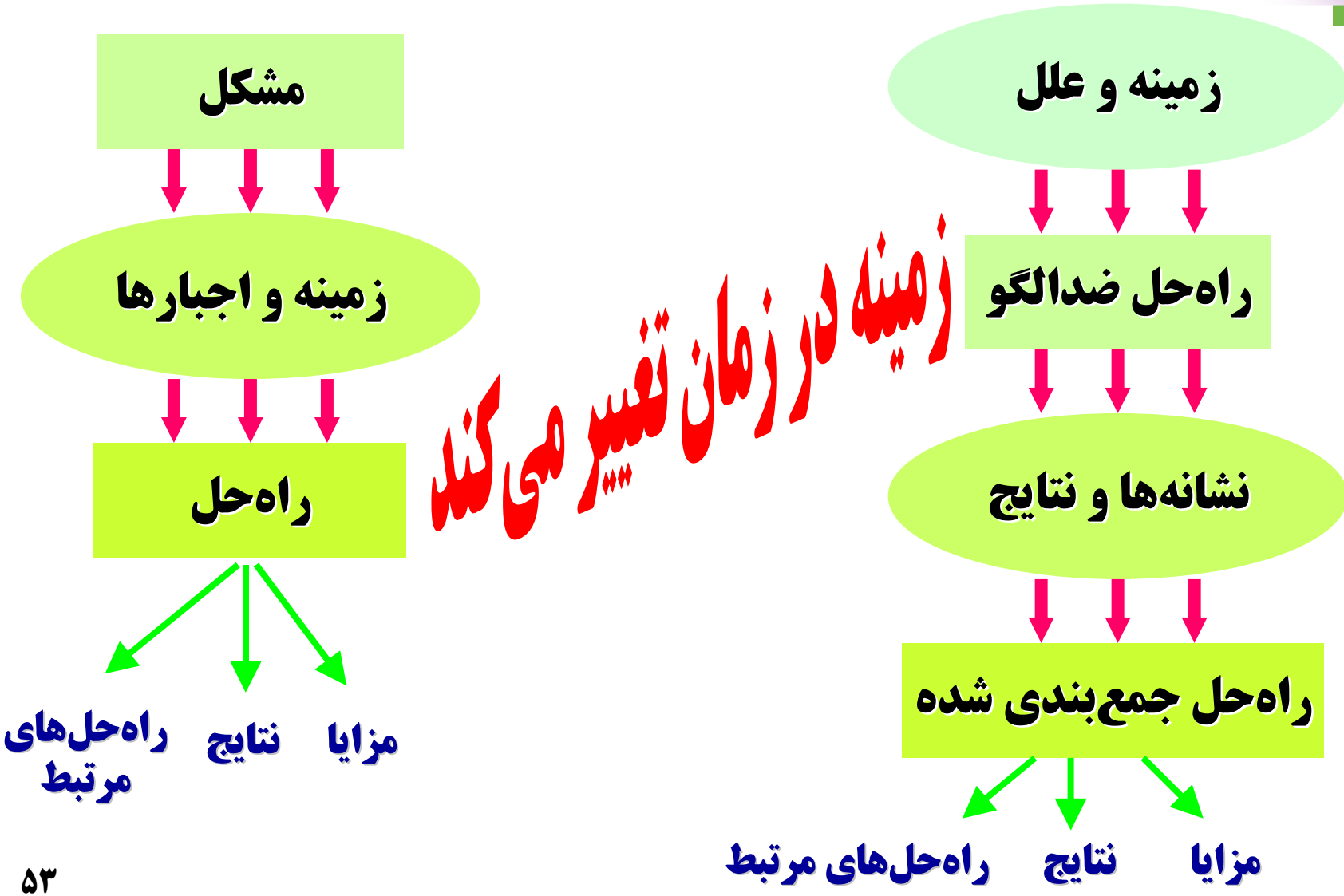
- بحث امروز
  - فناوری مولفه‌ها
  - اشیا توزیع‌شده
  - معماری سرویس‌گرا
  - الگوها
  - استفاده مجدد از نرم‌افزار
  - عامل‌های نرم‌افزار (*Agents*)
  - رابط *Web*

رابطه بین الگو و ضدالگو

- اغلب الگوهای طراحی منجر به ضدالگو می‌شوند
  - برنامه‌نویسی رویه‌ای در دهه ۶۰ و ۷۰ یک الگوی طراحی بـود، امـا در حال حاضر ضدالگو است
  - شی‌گرایی در حال حاضر یک الگو است، اما در حال تبدیل به ضـدالگو است



رابطه بین الگو و ضدالگو (ادامه)



نمونه ضدالگوها

- *The Blob*
- *Continuous obsolescence*
- *Lava Flow*
- *Ambiguous viewpoint*
- *Functional decomposition*
- *Poltergeists*
- *Boat Anchor*

- *Golden Hammer*
- *Dead End*
- *Spaghetti Code*
- *Input Kludge*
- *Walking through a Minefield*
- *Cut-and-Paste Programming*
- *Mushroom Management*

طبقه‌بندی ضدالگوها

- ضدالگوهای توسعه
- ضدالگوهای معماری
- ضدالگوهای مدیریتی

ضدالگوهای توسعه – *Lava Flow*

- **مشکل**
  - استفاده از کد قدیمی و اطلاعات طراحی فراموش‌شده در طراحی دایم‌التغییر
- **نمونه**
  - طراح ارشد استعفا نموده است
  - طراح جدید روش بهتری ارائه نموده است، اما بدلیل عدم آشنایی با کد نمی‌تواند آنها را حذف نماید

ضدالگوهای توسعه – *Lava Flow*

- **علل**
  - توزیع کنترل‌نشده کدهای اتمام نشده یا اصلاح نشده
  - قسمت‌ها و مسیرهایی که برای تست در کد قرار داده شده بودند، حذف نشدند
  - شکاف در معماری بخاطر استفاده از فناوری قدیمی
    - شکاف در معماری وقتی بوجود می‌آید که معماری اولیه پس از شروع توسعه دچار تغییرات شگرف شود
  - فقدان معماری
  - فقدان مدیریت پیکربندی

ضدالگوهای توسعه – *Lava Flow*

- **راه‌حل**

- استفاده از سیستم مدیریت پیکربندی سبب می‌شود که کدهای کهنه شناسایی و حذف شوند
- تکمیل‌نمودن طراحی (قدیمی)
- استفاده از یک معماری سالم برای پیش‌بردن مراحل توسعه
- استفاده از رابط در سطح سیستم که خوش-تعریف، پایا و بدرستی مستند شده باشد

ضدالگوهای توسعه – *Blob*

- **مشکل**
  - استفاده از سبک طراحی رویه‌ای منجر به ایجاد موضوعی با مسئولیت بسیار می‌شود
  - بقیه موضوعات تنها داده نگهداری می‌نمایند
  - این کلاس همان کلاسی است که قلب معماری خواهد بود
  - این کلاس پردازش و دیگر داده‌ها را انحصاری می‌نماید

ضدالگوهای توسعه – *Blob*

- **علل**
  - فقدان دیدگاه معماری شی‌گرایی
  - عدم وجود التزام به معماری
  - طراح رویه‌ای معمار ارشد است
  - عدم مداخله در طراحی کلاس‌ها
    - در پروژه‌های تکراری افراد سعی می‌کنند کار خود را افزایش ندهند و ترجیح می‌دهند از کلاس‌های قدیمی‌تر استفاده نمایند، بدون اینکه نیاز به نوآوری داشته باشند

ضدالگوهای توسعه – *Blob*

- راه‌حل
  - توزیع مجدد وظیفه‌مندی
  - جداسازی اثر تغییرات
  - تعیین یا طبقه‌بندی خصوصیات و عملیات
  - حذف «ارتباطات بسیار دور»، زائد یا غیرمستقیم
  - حذف تمام ارتباطات موقت