معرفی الگوهای طراحی

در مهندسی نرم‌افزار، الگوی طراحی یک راه‌حل عمومی قابل تکرار برای مشکلات متداول در زمینه طراحی نرم‌افزار است. الگوی طراحی، یک طراحی تمام‌شده نیست که به صورت مستقیم بتواند تبدیل به کد منبع یا ماشین شود؛ بلکه، یک توضیح یا قالب برای حل یک مسئله در شرایط مختلف است. الگوها در واقع بهترین روش ممکن هستند که یک برنامه‌نویس می‌تواند در هنگام طراحی یک برنامه برای حل مشکلاتش از آن‌ها استفاده کند. الگوهای طراحی شیءگرا نوعاً نشان‌دهندهٔ روابط و تعامل‌ها بین کلاس‌ها و شیءها هستند، بدون این‌که کلاس‌ها یا اشیا نهایی برنامه را مشخص کند. الگوهایی که در خود وضعیت‌های تغییرپذیر دارند، شاید مناسب زبان‌های برنامه‌نویسی تابعی نباشند. هم‌چنین، در بعضی از زبان‌ها که برای حل یک مسئله راه‌حل‌های آمادهٔ از پیش تعریف‌شده وجود دارد، استفاده از بعضی الگوها برای حل مسئلهٔ مشابه می‌تواند لازم نباشد. به همین ترتیب، الگوهای طراحی شیءگرا ممکن است برای زبان‌های غیر شیءگرا مناسب نباشند.

به طور کلی ۲۳ الگوی طراحی کلاسیک، در ۳ گروه زیر مطرح شده اند :

۱-الگوهای ایجادی

- (Abstract factory) رابطی برای ساخت فامیلی‌هایی از اشیاء مرتبط یا وابسته به هم بدون مشخص کردن کلاس‌های واقعی آن‌ها تدارک می‌بیند.
- (Builder) فرآیند ساخت یک شیء را از نمایش آن جدا کرده تا بتوان همان فرآیند ساخت را برای ایجاد نمایش‌های مختلف بکار برد.
- (Factory method) یک رابط برای ساخت اشیاء تعریف کرده و در عین حال اجازه می‌دهد تا زیر کلاس‌ها در مورد اینکه چه شیء را نمونه‌سازی کنند تصمیم بگیرند. در واقع این الگو اجازه می‌دهد تا نمونه‌سازی را به زیر کلاس‌ها محول نمود.
- (Lazy Initialization) شیوهٔ تأخیر انداختن برای ساخت یک شئ، محاسبه یک مقدار یا پردازش‌های سنگین دیگر تا زمانِ اولین نیاز به آن‌ها. این الگو در فهرست GoF با نام «پروکسی مجازی» نیز شناخته می‌شود. یک استراتژی پیاده‌سازی برای الگوی Proxy به حساب می‌آید.
- (Multiton) این اطمینان را حاصل می‌کند که یک کلاس فقط نمونه‌های (instances) با نام دارد و یک نقطه سراسری (global) برای دسترسی به آن فراهم می‌کند.
- (Object pool) با بازیابی اشیائی که دیگر مورد استفاده قرار نمی‌گیرند از اشغال و آزادسازی‌های سنگین منابع دوری می‌کند.
- (Prototype) با استفاده از یک شیء به عنوان نمونه نوع اشیاء جدیدی که بایستی ساخته شوند را مشخص کرده و آن اشیاء را با ساختن کپی‌های جدید از این نمونه ایجاد می‌نماید.

- (Resource acquisition is initialization) این اطمینان را حاصل می‌کند که منابع به‌طور مناسب با تعیین طول عمر برای اشیاء آزادسازی می‌شوند.
- (Singleton) این اطمینان را حاصل می‌کند که یک کلاس دارای تنها یک نمونه بوده و دسترسی به آن نمونه را تدارک می‌بیند.
- (Object library) کپسوله کردن مدیریت اشیاء شامل factory interface و لیست‌های مُرده و زنده

۲-الگوهای ساختاری

- (Adapter) رابط یک کلاس را به رابط دیگری که مورد انتظار یک مشتری است تبدیل می‌کند. این الگو امکان همکاری بین اشیائی که قبلاً بخاطر داشتن رابط‌های ناسازگار نمی‌توانستند با هم کار کنند را فراهم می‌سازد.
- (Bridge)یک مفهوم مجرد را از پیاده‌سازی اش مجزا کرده تا هردو بتوانند به‌طور مستقل تغییر کنند.
- (Composite) اشیاء را به صورت ساختار درختی برای ایجاد ساختار سلسله مراتبی بفرم-part whole ترکیب می‌نماید. این الگو اجازه می‌دهد تا مشتری‌ها اشیاء منفرد و مرکب را به صورت یکسانی پردازش کنند.
- (Decorator)در زمان اجرا وظایف جدیدی به یک شی ء اضافه می‌کند. این الگو بدیلی قابل انعطاف برای گسترش عملکرد یک کلاس به وسیلهٔ زیر کلاس ساختن از آن را فراهم می‌کند.
- (Facade)یک رابط منفرد برای مجموعه‌ای از رابط‌ها در یک زیر سیستم تدارک می‌بیند. در واقع یک facade رابط سطح بالاتری برای یک زیر سیستم تعریف کرده و باعث می‌شود تا زیر سیستم را به صورت ساده تری مورد استفاده قرار داد.
- (Flyweight)از اشتراک منابع برای فراهم نمودن تعداد زیادی از اشیاء سبک به صورت کارا استفاده می‌کند.
- (Proxy)یک جانشین یا جایگاه برای کنترل دسترسی به یک شیء ایجاد می‌کند.

۳-الگوهای رفتاری

- (Chain of responsibility)با دادن شانس به بیش از یک شیء برای پاسخگویی به یک درخواست از در هم آمیختن فرستنده و دریافت‌کننده یک درخواست جلوگیری می‌کند. به این ترتیب که اشیاء دریافت‌کننده یک درخواست را به صورت زنجیره‌ای در نظر گرفته و درخواست را در طول این زنجیره عبور داده تا اینکه یکی از اشیاء آن را پاسخ پوید.
- (Command)یک درخواست را به صورت یک شیء کپسول‌سازی می‌کند. بنابراین این امکان را فراهم می‌کند تا مشتری‌ها را با درخواستهای متفاوت پارامتردهی کرده، درخواستها را صف بندی یا Log کرده و اعمال قابل برگشت فراهم کنید.

- (Interpreter)برای یک زبان، نمایشی از گرامرش به همراه مفسری که از آن نمایش برای تفسیر جملات مربوط به آن زبان استفاده می‌کند تعریف می‌نماید.
- (Iterator)روشی برای دسترسی ترتیبی به عناصر یک شیء مجتمع (مرکب) بدون افشاء کردن نمایش آن فراهم می‌کند.
- (Mediator)شیءی که نحوه تبادلات مجموعه‌ای از اشیاء را کپسول‌سازی می‌کند را تعریف می‌نماید. این الگو با پرهیز از ارجاعات مستقیم بین مجموع‌های از اشیاء، اتصال حداقلی بین آن‌ها را ترغیب نموده و اجازه می‌دهد تا تبادلات را به صورت مستقل تغییر دهید.
- (Memento)بدون شکستن کپسول سازی، حالت درونی یک شیء را تسخیر و ذخیره کرده تا آن شیء بتواند بعداً به آن حالت برگشت یابد.
- (Observer)یک نوع وابستگی یک- به- چند بین اشیاء تعریف کرده بطوری‌که وقتی یک شیء حالتش را تغییر داد تمام اشیاء وابسته به آن خبردار شده تا آن‌ها خود را با آن تغییر هماهنگ کنند.
- (State)اجازه می‌دهد تا یک شیء هنگامیکه حالتش عوض شد رفتارش را تغییر دهد. اشیاء از این نوع رفتار کلاسی که در آن قرار دارند را تغییر می‌دهند.
- (Strategy)یک خانواده از الگوریتم‌ها را تعریف؛ کپسول‌سازی و قابل جایگزین کردن می‌کند. استراتژی اجازه می‌دهد تا یک الگوریتم را بدون توجه به جائیکه مورد استفاده قرار می‌گیرد تغییر داد.
- (Template method)اسکلت یک الگوریتم را تعریف کرده و پیاده‌سازی بعضی قدم‌های آن را به زیر کلاس‌ها محول می‌کند. این الگو امکان تغییر بعضی از قدم‌های یک الگوریتم را بدون تغییر در ساختار کلی الگوریتم به زیر کلاس‌ها می‌دهد.
- (Visitor)عملی که بایستی بر روی عناصر یک ساختار از اشیاء اعمال شود را نمایش می‌دهد. این الگو اجازه می‌دهد تا عمل جدیدی بدون نیاز به تغییر کلاس‌های عناصری که بر روی آن عمل می‌کند را تعریف کنید.

امیرمهدی مختاری(۹۸۳۱۱۴۳)

Resources:

1. https://sourcemaking.com/design_patterns
2. https://en.wikipedia.org/wiki/Software_design_pattern
3. https://refactoring.guru/design-patterns
4. https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. https://www.geeksforgeeks.org/software-design-patterns/
6. https://web.archive.org/web/20080229011119/http://developer.yahoo.com/ypatterns/
7. https://torgronsund.wordpress.com/2010/01/06/lean-startup-business-model-pattern/