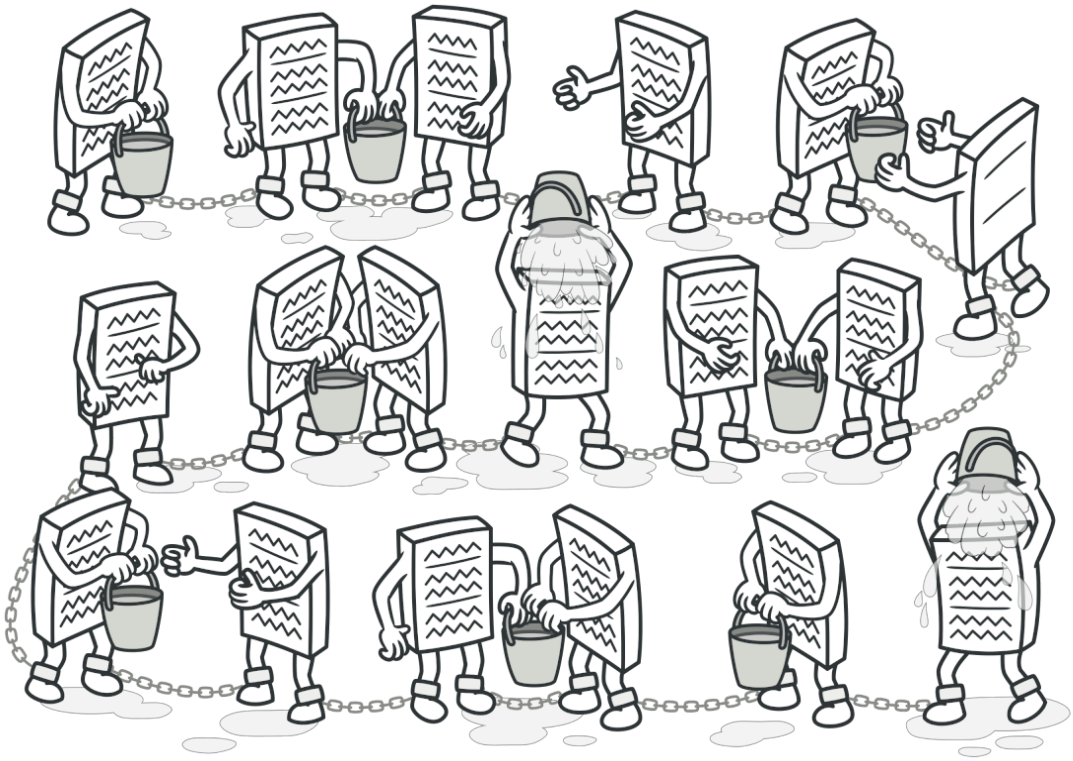
Chain of Responsibility

Behavioral design pattern



Nona Ghazizadeh

Shayan Mohammadizadeh

مقدمه

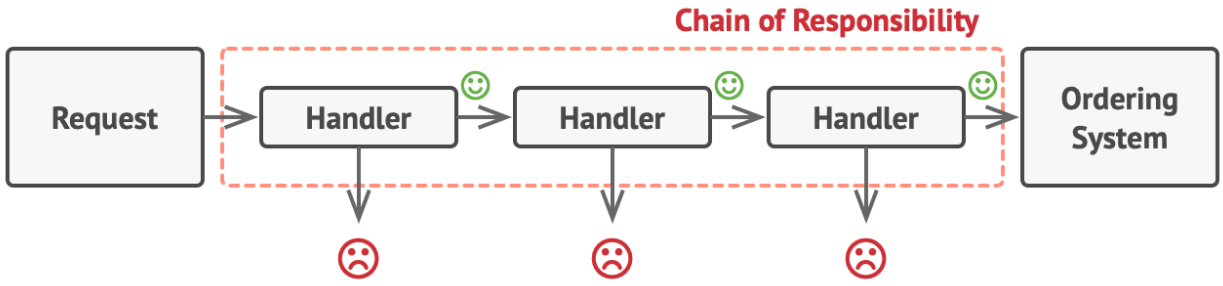
الگوی طراحی chain of responsibility که در لغت به معنی زنجیره مسئولیت‌ها است. راه‌حلی برای کارهای سلسله مراتبی با خوانایی بالای کد است. کارهای سلسله مراتبی، کارهایی هستند که بررسی یک درخواست توسط یک بخش وابسته به تصمیم گرفته شده و نتیجه به دست آمده در بخش پیشین است به عبارتی دیگر، به محض ورود یک درخواست هر یک از بخش‌های این زنجیره تصمیم می‌گیرند که آن را به بخش بعدی بفرستند یا در همان بخش آن در خواست را بررسی کند.

کاربرد

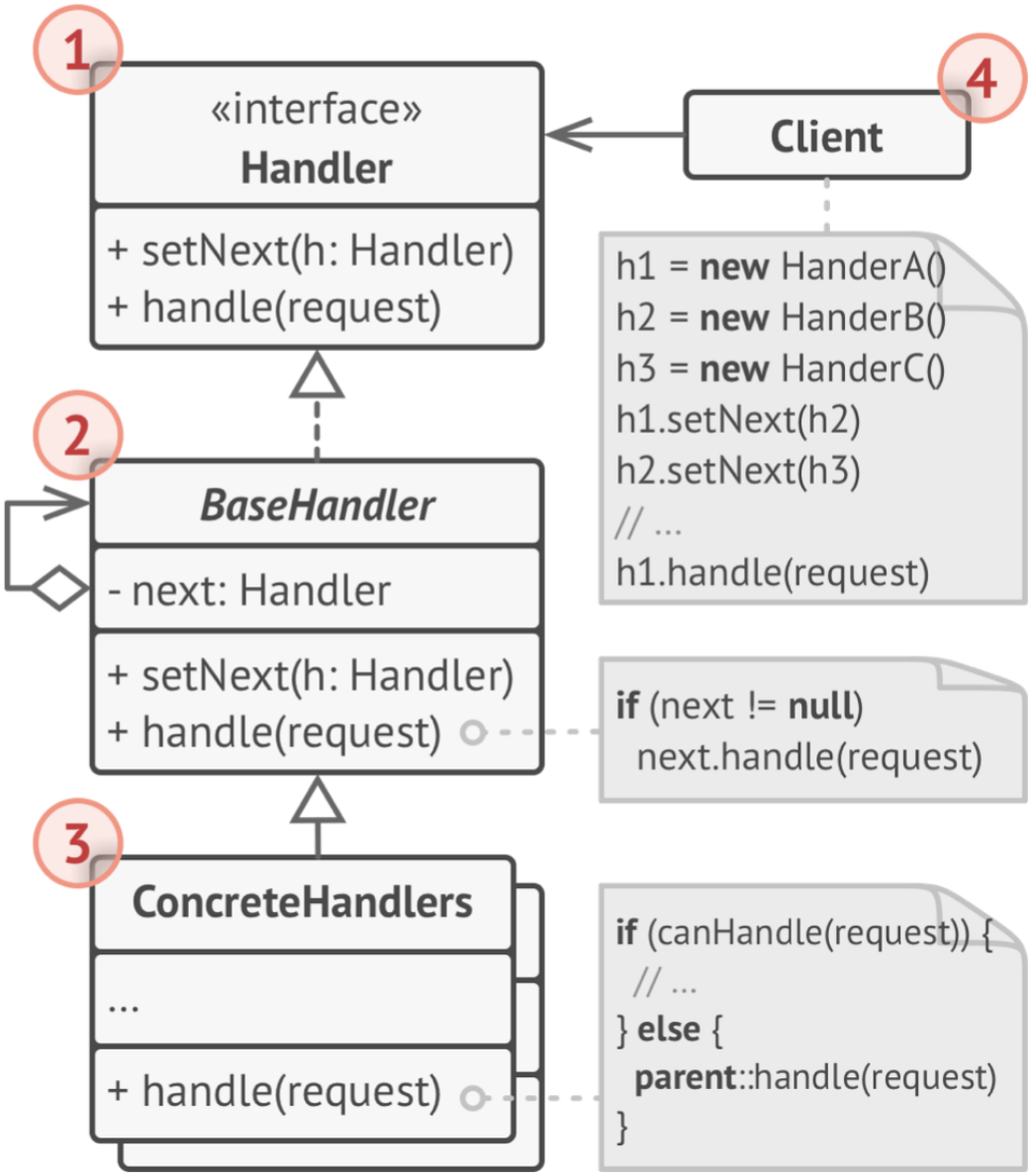
فرض کنید می‌خواهید انتقال کارت به کارت را از طریق تماس تلفنی بررسی کنید. به طوری که ابتدا یک تماس تلفنی گرفته می شود از شما می‌خواهد کد ملی خود را که با شماره کارت و تلفنی که با آن تماس گرفته‌اید را وارد کنید سپس در صورت تطبیق از شما شماره کارت طرف مقابل را می خواهد با وارد کردن شماره کارت اسم شخص را برای شما می‌گوید و از شما درخواست تأیید یا رد می‌کند در صورت تایید شما از شما مبلغ واریزی را می‌خواهد و مبلغ وارد شده را وارد می‌کنید و عملیات به پایان می‌رسد، حال با توجه به فرآیندی که گفته شد با توجه به مبانی الگوریتمی، راه ساده‌ای که به ذهن می‌رسد استفاده از if-elseif-else یا همان switch-case است. اما فرض کنید از شما بخواهند قابلیت‌ها و بررسی محدودیت‌های بیشتری را به این سیستم اضافه کنید به طور مثال در صورتی که کاربر بیش از سه بار با همان شماره تلفن، کد ملی که تطبیق ندارد را وارد کند تماس تلفنی را قطع کرده و تا زمان احراز هویت فرد به یکی از شعب بانک کارت او مسدود خواهد بود و یا از ما می‌خواهند تا هنگام وارد کردن مبلغ برای انتقال وجه، سقف آن را بررسی کنیم، در صورتی که از سقف آن بیشتر بود عملیات پایان نمی‌یابد. همچنین ممکن است یک بانک علاوه بر انتقال وجه بخواهد کارهای بانکی دیگر را نیز از طریق تلفن میسر سازد بنابراین هنگام برقراری تلفن، تلفن گویا عملیات مختلف را با شماره‌های مختلف برای کاربر خوانده و در صورت انتخاب هر یک وارد پردازش آن بخش می‌شود.

واضح است که استفاده از عبارات شرطی همان if-elseif-else یا switch-case کد را ناخوانا کرده و جز best practice ها نمی‌باشد .

برای خوانایی کد و پیاده‌سازی موارد خواسته‌شده از آبجکت‌های stand-alone که به آنها handler می‌گوییم استفاده می‌کنیم بدین صورت که این handler ها را مانند linked list به هم متصل کنیم. برای پردازش یک درخواست زمانی موفق خواهیم شد که تمام handler ها امکان پردازش درخواست را داشته باشند. هر یک از بررسی محدودیت‌ها باید در کلاس مخصوص همان handler توسط یک تابع که آرگومان ورودی آن همان درخواست ما است که این دقیقاً همان الگو chain of responsibility است.



ساختار



1. handler یک interface برای تمام handler ها است که شامل دو تابع، یکی برای هندل کردن درخواست، دیگری برای مشخص کردن handler بعدی است
2. کلاس BaseHandler را برای جلوگیری از کد تکراری در کلاس‌های handler پیاده‌سازی می‌کنیم. یک field برای ارجاع دادن به handler بعدی خود دارد دو تابع مشابه همان چیزی که پیشتر گفته شد دارد.
3. کلاس ConcreteHandlers بخش پردازش درخواست را انجام می‌دهد. هنگام ورود یک درخواست، هر handler باید تصمیم بگیرد که پردازش را انجام دهد یا به handler بعدی منتقل کند.

4. با توجه به منطق پیاده‌سازی شده برای برنامه کاربر می‌تواند زنجیره‌ای از handler ها تنها یکبار بسازد و یا به طور پویا بسازد.

سودو کد

```
// an interface for handlers

Interface Handler:
   setNext(handler: Handler): Handler;
   handle(request: String): string;

// implement parent class for handlers

Class BaseHandler implements Handler:
   nextHandler: Handler;

   Func setNext(handler: Handler):
      this.nextHandler = handler;
      Return this.nextHandler

   Func handle(request: string):
      If (this.nextHandler):
         Return this.nextHandler.handle(request)

      Return null;

// implement classes for each of our handlers

Class AHandler extends BaseHandler:
   Func handle(request: string):
      // ...

Class BHandler extends BaseHandler:
   Func handle(request: string):
      // ...

Class CHandler extends BaseHandler:
   Func handle(request: string):
      // ...
```

```
Func client(handler: Handler):
   Const request = ...
   handler.handle(request)
```

خوبی(ها) و بدی(ها)

😃 می‌توان ترتیب درخواست‌ها را مشخص کرد که کدام عمل پیش از دیگری و کدام بعد از دیگری انجام شود.

😃 می‌توان handler جدیدی را در سیستم بدون از ابتدا کد زدن کد موجود تعریف کرد.

😃 این قاعده که کلاس‌ها، هر کدام فقط مسئول یک قسمت منطقی از سیستم هستند را رعایت می‌کند.

😢 این احتمال وجود دارد که برخی از درخواست‌ها بدون پردازش بمانند.