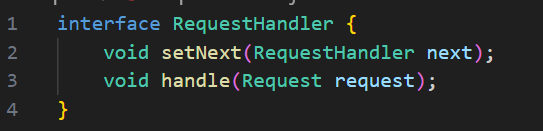
**Case study 2: HTTP request handling system based on Decorator design pattern**

In Web services, an HTTP request typically goes through the following steps: authentication, logging, data validation, and logic execution. In order to avoid coupling all functions in a class, the system adopts decorator design pattern, which encapsulating each function into an independent module, and realizing the dynamic extension of functions through structural combination.

The structure of this system is consistent with the decorator model, which mainly includes the following five types of roles：

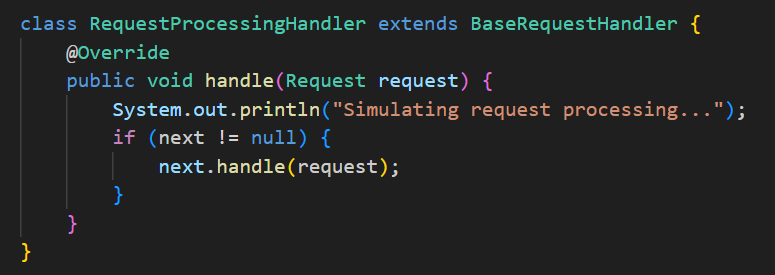
**1. Component (interface) -- RequestHandler**

It defines a uniform handling method, handle(), and a chain combination method, setNext(), to provide specifications for all processor classes. It is the core contract of the pattern, ensuring consistent behavior between the base components and decorators.



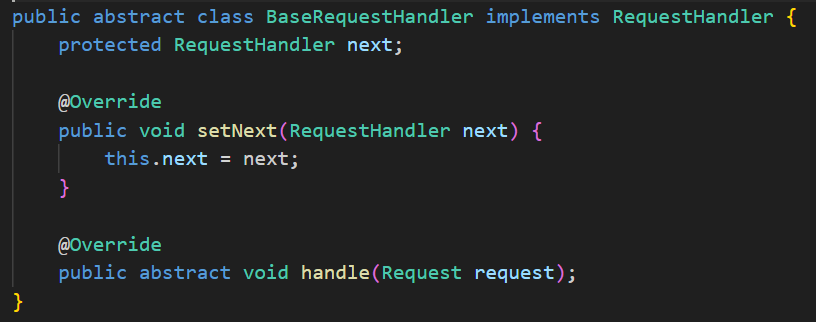
**2. Concrete Component —— RequestProcessingHandler**

It is the ultimate executor of request processing, which encapsulating the core execution logic without any additional behavior. It can be altered by all decorators, reflecting the essential role of the "decorator".



**3. Base Decorator —— BaseRequestHandler**

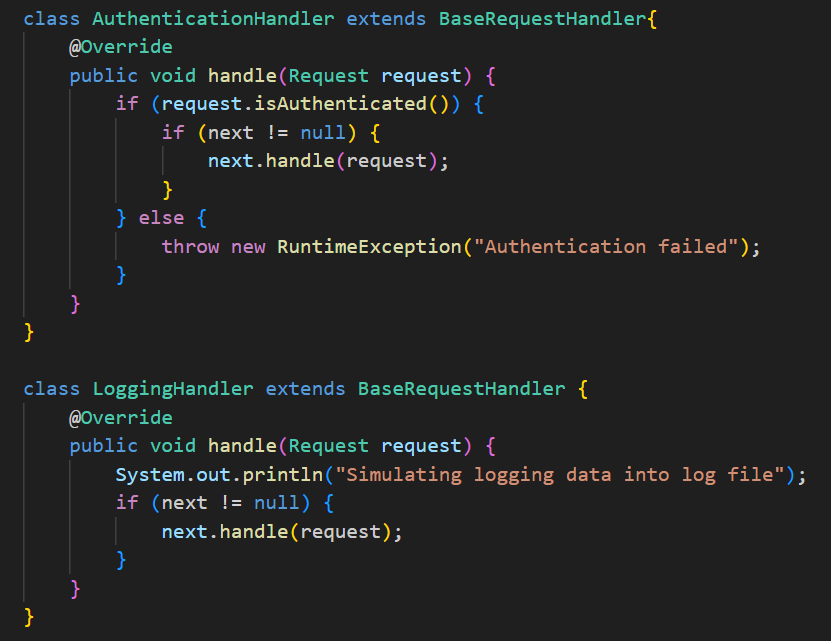
As an abstract base class, it implements RequestHandler interface and holds the next handler reference. All concrete decorator classes inherit from this class and forward the request through next.handle().



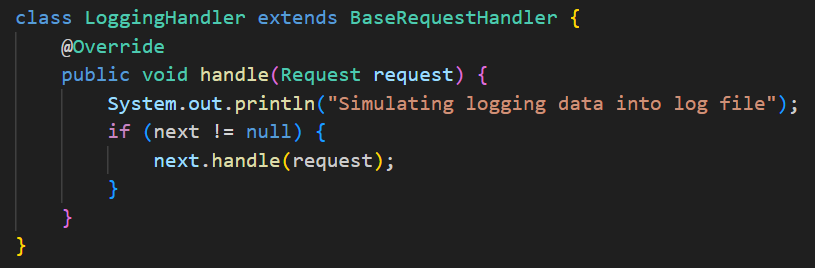
**4. Concrete Decorators**

These classes dynamically add their own functional logic when processing the request, reflecting the "functional wrapping" in the decorator pattern：

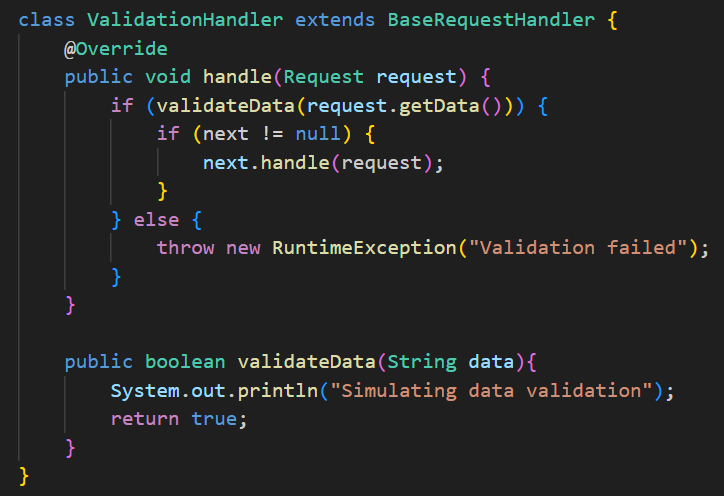
* AuthenticationHandler：Authentication



* LoggingHandler：Log recording



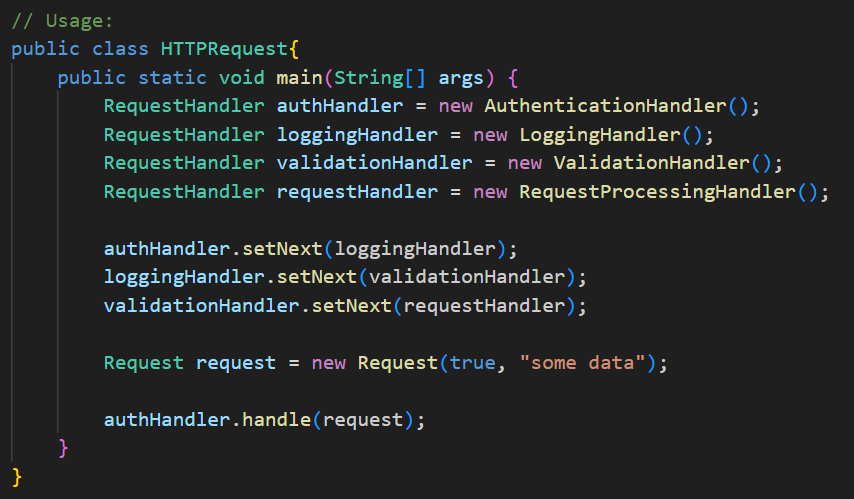
* ValidationHandler：Request parameter validation



Each processor calls next.handle() after executing its own logic, nesting functions layer by layer.

**5. Client —— HTTPRequest.main()**

It builds a request-handling chain by instantiating the concrete derectors and connecting them in turn using the setNext() method. The call chain starts with the outermost processor and passes layer by layer to the core processing logic. Without modifying existing classes, this structure realizes the flexible extension of functions.



Overall, the system implementing the core idea of decorator pattern through unified interface and combinable function modules. Compared to inheritance or hard coding, it is more flexible, extensible, and decoupled for flexible combinations of complex functions.

In No Silver Bullet, Brooks points out that software development faces four fundamental difficulties: complexity, changeability, conformity, and invisibility, which cannot be solved by one technology at a time. By encapsulating functions into independently composable decorative classes, decorator design pattern reduces the coupling of core logic, improves module clarity and maintainability. It also helps to cope with the difficulties of understanding and expansion caused by complexity. When requirements change, developers can add or replace decorators as needed to enhance the flexibility and adaptability of the system. At the same time, the way of chained composition also helps to solve the abstract structure, alleviating the problem of invisibility. While not totally eradicating the underlying challenge, the decorator pattern provides a structured, scalable solution.