Chain of responsibility

Objective

Avoid coupling the class requesting an action with the class executing it to allow more than one object to handle the request. Chain the receiving objects and pass the request along the whole chain until one object handles it.

Function

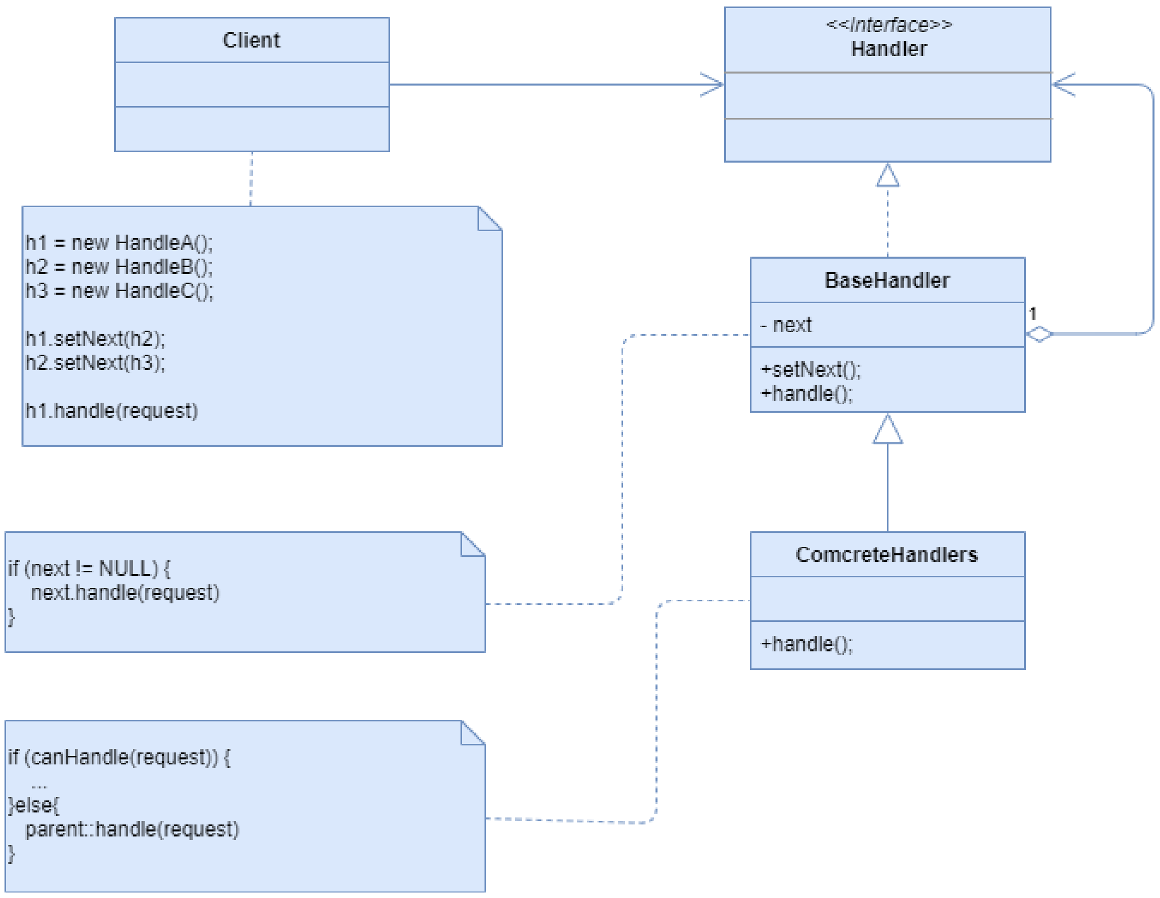
Establish the line that must carry the messages for the objects to perform the indicated task.

Structure

As shown in figure 1

The Handler class declares the interface, common to all ConcreteHandlers classes. It usually contains only one method for handling requests, but sometimes it can also have another method for setting up the next handler in the chain.

The structure that meets this pattern is shown in Figure 1



```
h1 = new HandleA();
h2 = new HandleB();
h3 = new HandleC();

h1.setNext(h2);
h2.setNext(h3);

h1.handle(request)
```

```
if (next != NULL) {
    next.handle(request)
}
```

```
if (canHandle(request)) {
    ...
}else{
   parent::handle(request)
}
```

Figure 1: UML Diagram Chain of responsibility Pattern

Applications

The use of the Chain of Responsibility pattern is recommended when:

- Several objects must handle a request, and determining who does it is not a priority since the object that handles it is determined automatically.

- You want to make a request to one of several objects without explicitly specifying who should receive the request.

- The set of objects capable of handling a request must be defined dynamically.

Design Patterns Collaborators

- The chain of responsibility pattern is usually implemented along with the composite pattern, as a parent component can be a successor.

Scope of action

Applied at the object level.

Problem

An application needs to execute an action independently of its state, conventionally it is possible to cover this requirement, but to achieve it, each object must instantiate the respective method that is in charge of executing the request and also know the objects capable of executing the same action and invoke them in a certain order, which generates a high coupling between objects.

Solution

The Chain of Responsibility design pattern presents a more optimal solution to cover this requirement, since it proposes to prepare each object to be able to handle a request on its own or to pass the request to another dynamically defined object at execution time, which generates a chain of objects that are potential executors of a request. If the request cannot be executed, the object passes it to the next link in the chain.

Diagram or Implementation

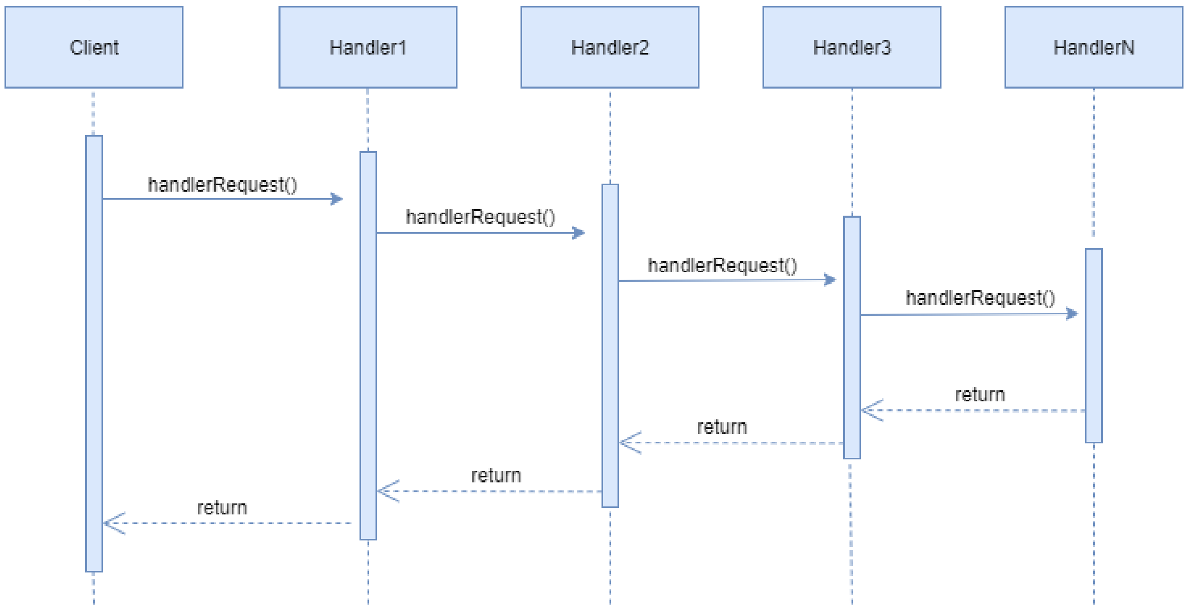


Figure 2: UML Diagram Chain of responsibility Pattern

Figure 2 explains the behaviour of the pattern by means of a sequence diagram.

- The client class requests the processing of a request to a chain of responsibility.

- The first Handler component tries to process the message, however, it is not able to process it for some reason and sends the message to the next handler component in the chain.

- The second Handler component tries to process the message without success, so it sends the message to the next Handler component in the chain.

- The third Handler component also tries to process the message without success and sends the message to the next Handler in the chain.

- The HandlerN component (Some sequence handler) is finally able to process the message successfully and returns a response (optional) so that the response is replicated by all the past Handler components until it reaches the client class.