Chain of Responsibility in Java

Chain of Responsibility is behavioral design pattern that allows passing request along the chain of potential handlers until one of them handles request.

The pattern allows multiple objects to handle the request without coupling sender class to the concrete classes of the receivers. The chain can be composed dynamically at runtime with any handler that follows a standard handler interface.

Learn more about Chain of Responsibility →

Navigation

Complexity: ★★☆

Usage examples: The Chain of Responsibility is pretty common in Java.

One of the most popular use cases for the pattern is bubbling events to the parent components in GUI classes. Another notable use case is sequential access filters.

Here are some examples of the pattern in core Java libraries:

- `javax.servlet.Filter#doFilter()`
- `java.util.logging.Logger#log()`

Identification: The pattern is recognizable by behavioral methods of one group of objects that indirectly call the same methods in other objects, while all the objects follow the common interface.

Filtering access

This example shows how a request containing user data passes a sequential chain of handlers that perform various things such as authentication, authorization, and validation.

This example is a bit different from the canonical version of the pattern given by various authors. Most of the pattern examples are built on the notion of looking for the right handler, launching it and exiting the chain after that. But here we execute every handler until there's one that **can't handle** a request. Be aware that this still is the Chain of Responsibility pattern, even though the flow is a bit different.

middleware

middleware/Middleware.java: Basic validation interface

```
package refactoring_guru.chain_of_responsibility.example.middleware;

/**
 * Base middleware class.
 */
public abstract class Middleware {
    private Middleware next;
```

```
     * Builds chains of middleware objects.
     */
    public static Middleware link(Middleware first, Middleware... chain) {
        Middleware head = first;
        for (Middleware nextInChain: chain) {
            head.next = nextInChain;
            head = nextInChain;
        }
        return first;
    }

    /**
     * Subclasses will implement this method with concrete checks.
     */
    public abstract boolean check(String email, String password);

    /**
     * Runs check on the next object in chain or ends traversing if we're in
     * last object in chain.
     */
    protected boolean checkNext(String email, String password) {
        if (next == null) {
            return true;
        }
        return next.check(email, password);
    }
}
```

middleware/ThrottlingMiddleware.java: Check whether the limit on the number of requests is reached

```
package refactoring_guru.chain_of_responsibility.example.middleware;

/**
 * ConcreteHandler. Checks whether there are too many failed login requests.
 */
public class ThrottlingMiddleware extends Middleware {
    private int requestPerMinute;
    private int request;
    private long currentTime;

    public ThrottlingMiddleware(int requestPerMinute) {
        this.requestPerMinute = requestPerMinute;
```

```java
    /**
     * Please, not that checkNext() call can be inserted both in the beginning
     * of this method and in the end.
     *
     * This gives much more flexibility than a simple loop over all middleware
     * objects. For instance, an element of a chain can change the order of
     * checks by running its check after all other checks.
     */
    public boolean check(String email, String password) {
        if (System.currentTimeMillis() > currentTime + 60_000) {
            request = 0;
            currentTime = System.currentTimeMillis();
        }

        request++;

        if (request > requestPerMinute) {
            System.out.println("Request limit exceeded!");
            Thread.currentThread().stop();
        }
        return checkNext(email, password);
    }
}
```

middleware/UserExistsMiddleware.java: Check user’s credentials

```java
package refactoring_guru.chain_of_responsibility.example.middleware;

import refactoring_guru.chain_of_responsibility.example.server.Server;

/**
 * ConcreteHandler. Checks whether a user with the given credentials exists.
 */
public class UserExistsMiddleware extends Middleware {
    private Server server;

    public UserExistsMiddleware(Server server) {
        this.server = server;
    }

    public boolean check(String email, String password) {
        if (!server.hasEmail(email)) {
        }
        if (!server.isValidPassword(email, password)) {
            System.out.println("Wrong password!");
            return false;
        }
        return checkNext(email, password);
    }
}
```

middleware/RoleCheckMiddleware.java: Check user’s role

```
package refactoring_guru.chain_of_responsibility.example.middleware;

/**
 * ConcreteHandler. Checks a user's role.
 */
public class RoleCheckMiddleware extends Middleware {
    public boolean check(String email, String password) {
        if (email.equals("admin@example.com")) {
            System.out.println("Hello, admin!");
            return true;
        }
        System.out.println("Hello, user!");
        return checkNext(email, password);
    }
}
```

server

server/Server.java: Authorization target

```
package refactoring_guru.chain_of_responsibility.example.server;

import refactoring_guru.chain_of_responsibility.example.middleware.Middleware;

import java.util.HashMap;
import java.util.Map;

/**
 * Server class.
```

```
    private Map<String, String> users = new HashMap<>();
    private Middleware middleware;

    /**
     * Client passes a chain of object to server. This improves flexibility and
     * makes testing the server class easier.
     */
    public void setMiddleware(Middleware middleware) {
        this.middleware = middleware;
    }

    /**
     * Server gets email and password from client and sends the authorization
     * request to the chain.
     */
    public boolean logIn(String email, String password) {
        if (middleware.check(email, password)) {
            System.out.println("Authorization have been successful!");

            // Do something useful here for authorized users.

            return true;
        }
        return false;
    }

    public void register(String email, String password) {
        users.put(email, password);
    }

    public boolean hasEmail(String email) {
        return users.containsKey(email);
    }

    public boolean isValidPassword(String email, String password) {
        return users.get(email).equals(password);
    }
}
```

Demo.java: Client code

```
package refactoring_guru.chain_of_responsibility.example;
```

```
import refactoring_guru.chain_of_responsibility.example.middleware.ThrottlingMiddleware;
import refactoring_guru.chain_of_responsibility.example.middleware.UserExistsMiddleware;
import refactoring_guru.chain_of_responsibility.example.server.Server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * Demo class. Everything comes together here.
 */
public class Demo {
    private static BufferedReader reader = new BufferedReader(new InputStreamReader(System.i
    private static Server server;

    private static void init() {
        server = new Server();
        server.register("admin@example.com", "admin_pass");
        server.register("user@example.com", "user_pass");

        // All checks are linked. Client can build various chains using the same
        // components.
        Middleware middleware = Middleware.link(
            new ThrottlingMiddleware(2),
            new UserExistsMiddleware(server),
            new RoleCheckMiddleware()
        );

        // Server gets a chain from client code.
        server.setMiddleware(middleware);
    }

    public static void main(String[] args) throws IOException {
        init();

        boolean success;
        do {
            System.out.print("Enter email: ");
            String email = reader.readLine();
            System.out.print("Input password: ");
            String password = reader.readLine();
            success = server.logIn(email, password);
        } while (!success);
    }
}
```

```
Enter email: admin@example.com
Input password: admin_pass
Hello, admin!
Authorization have been successful!


Enter email: wrong@example.com
Input password: wrong_pass
This email is not registered!
Enter email: wrong@example.com
Input password: wrong_pass
This email is not registered!
Enter email: wrong@example.com
Input password: wrong_pass
Request limit exceeded!
```

RETURN

← Proxy in Java

READ NEXT

Command in Java →

Chain of Responsibility in Other Languages

Home Refactoring Design Patterns Premium Content
Forum Contact us