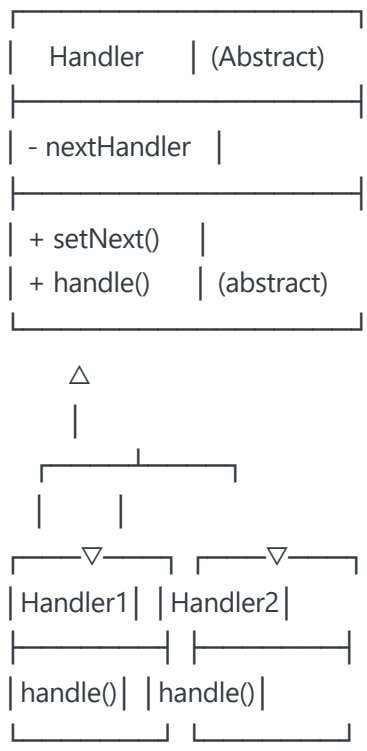
Chain of Responsibility Pattern

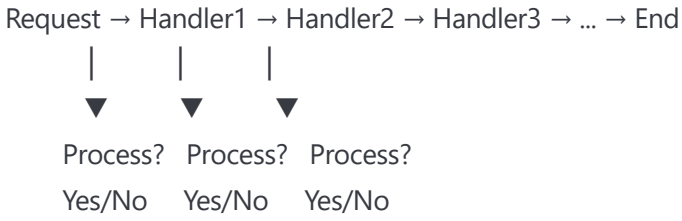
Concept

The Chain of Responsibility pattern passes requests along a chain of handlers. Each handler decides either to process the request or pass it to the next handler in the chain.

UML Class Diagram



Flow Diagram



Java Implementation

Abstract Handler

java

```
abstract class ExpenseHandler {
    protected ExpenseHandler nextHandler;

    public void setNext(ExpenseHandler handler) {
        this.nextHandler = handler;
    }

    public abstract void approveExpense(String expense, double amount);
}
```

Real-World Handlers

java

```
// Team Lead - handles small expenses
class TeamLeadHandler extends ExpenseHandler {
    @Override
    public void approveExpense(String expense, double amount) {
        if (amount <= 1000) {
            System.out.println("Team Lead approved: " + expense + " ($" + amount + ")");
        } else if (nextHandler != null) {
            nextHandler.approveExpense(expense, amount);
        }
    }
}

// Manager - handles medium expenses
class ManagerHandler extends ExpenseHandler {
    @Override
    public void approveExpense(String expense, double amount) {
        if (amount <= 5000) {
            System.out.println("Manager approved: " + expense + " ($" + amount + ")");
        } else if (nextHandler != null) {
            nextHandler.approveExpense(expense, amount);
        }
    }
}

// Director - handles large expenses
class DirectorHandler extends ExpenseHandler {
    @Override
    public void approveExpense(String expense, double amount) {
        if (amount <= 20000) {
            System.out.println("Director approved: " + expense + " ($" + amount + ")");
        } else if (nextHandler != null) {
            nextHandler.approveExpense(expense, amount);
        }
    }
}

// CEO - handles very large expenses
class CEOHandler extends ExpenseHandler {
    @Override
    public void approveExpense(String expense, double amount) {
        if (amount <= 100000) {
            System.out.println("CEO approved: " + expense + " ($" + amount + ")");
        } else {
            System.out.println("Expense requires board approval: " + expense + " ($" + amount + ")");
        }
```

```
    }
}
```

Client Usage

java

```
public class ExpenseApprovalSystem {
    public static void main(String[] args) {
        // Create approval chain
        ExpenseHandler teamLead = new TeamLeadHandler();
        ExpenseHandler manager = new ManagerHandler();
        ExpenseHandler director = new DirectorHandler();
        ExpenseHandler ceo = new CEOHandler();

        // Build the chain: TeamLead → Manager → Director → CEO
        teamLead.setNext(manager);
        manager.setNext(director);
        director.setNext(ceo);

        // Submit various expenses
        teamLead.approveExpense("Office supplies", 500);      // Team Lead
        teamLead.approveExpense("New laptop", 2500);          // Manager
        teamLead.approveExpense("Conference booth", 15000);   // Director
        teamLead.approveExpense("Office renovation", 75000);  // CEO
        teamLead.approveExpense("Building purchase", 500000); // Board approval
    }
}
```

Alternative: HTTP Middleware Example

java

```
abstract class HTTPMiddleware {
    protected HTTPMiddleware next;

    public void setNext(HTTPMiddleware middleware) {
        this.next = middleware;
    }

    public abstract boolean handle(HTTPRequest request);
}

class AuthenticationMiddleware extends HTTPMiddleware {
    @Override
    public boolean handle(HTTPRequest request) {
        if (!request.hasValidToken()) {
            System.out.println("Authentication failed");
            return false;
        }
        System.out.println("User authenticated");
        return next != null ? next.handle(request) : true;
    }
}

class AuthorizationMiddleware extends HTTPMiddleware {
    @Override
    public boolean handle(HTTPRequest request) {
        if (!request.hasPermission()) {
            System.out.println("Access denied - insufficient permissions");
            return false;
        }
        System.out.println("User authorized");
        return next != null ? next.handle(request) : true;
    }
}

class RateLimitMiddleware extends HTTPMiddleware {
    @Override
    public boolean handle(HTTPRequest request) {
        if (request.exceedsRateLimit()) {
            System.out.println("Rate limit exceeded");
            return false;
        }
        System.out.println("Rate limit check passed");
        return next != null ? next.handle(request) : true;
    }
}
```

Advantages

- **Loose Coupling**: Client doesn't need to know which handler will process the request
- **Dynamic Chain Building**: Handlers can be added, removed, or reordered at runtime
- **Single Responsibility**: Each handler focuses on one specific type of request
- **Open/Closed Principle**: Easy to add new handlers without modifying existing code
- **Flexibility**: Different chains can be built for different scenarios

Disadvantages

- **Performance Overhead**: Request may traverse multiple handlers before being processed
- **No Guarantee of Handling**: Request might reach end of chain without being processed
- **Debugging Complexity**: Harder to trace which handler will process a request
- **Runtime Configuration**: Chain structure is determined at runtime, making it less predictable
- **Memory Usage**: All handlers in chain must be kept in memory even if rarely used

Common Use Cases

- **GUI Event Handling**: Mouse clicks, keyboard events
- **Authentication**: Multiple auth methods (token, session, API key)
- **Logging**: Different log levels (debug, info, warning, error)
- **HTTP Middleware**: Request processing pipeline
- **Approval Workflows**: Multi-level authorization systems

Sequence Diagram

```
Client → Handler1: request
Handler1 → Handler1: canHandle?
alt can handle
    Handler1 → Client: process & return
else cannot handle
    Handler1 → Handler2: forward request
    Handler2 → Handler2: canHandle?
    alt can handle
        Handler2 → Client: process & return
    else cannot handle
        Handler2 → Handler3: forward request
    end
end
```