# Chain of Responsibility

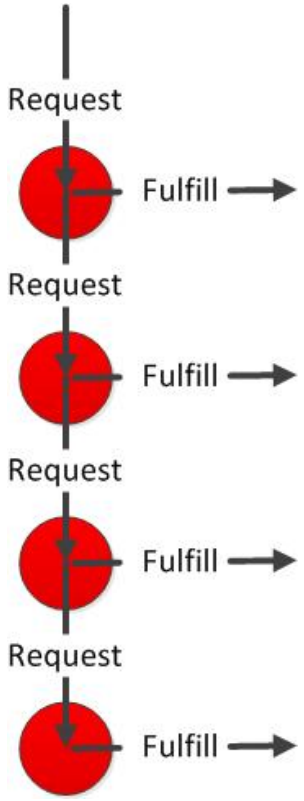
A behavioral pattern

# Learning goals

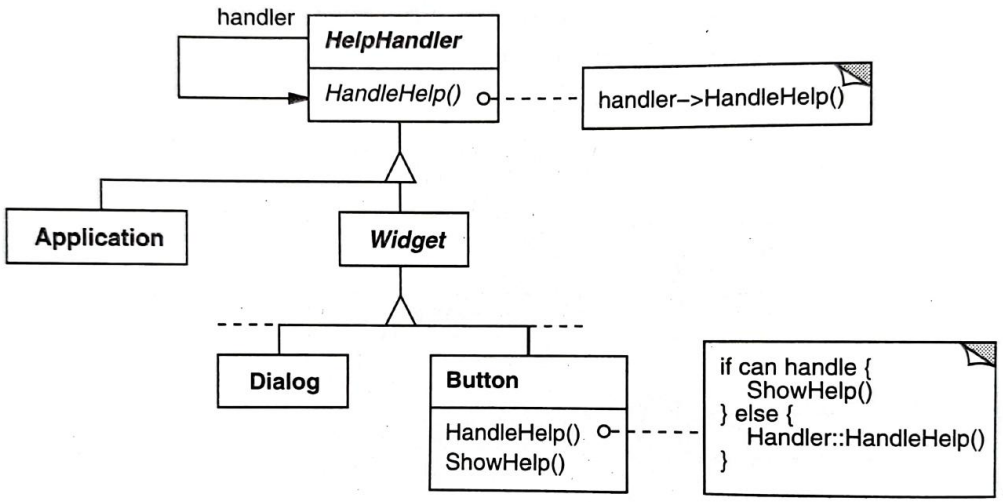
1. Learn the idea, structure, and Java implementation of the Chain of Responsibility design pattern.
2. Learn to apply the Chain of Responsibility DP in your own programming.

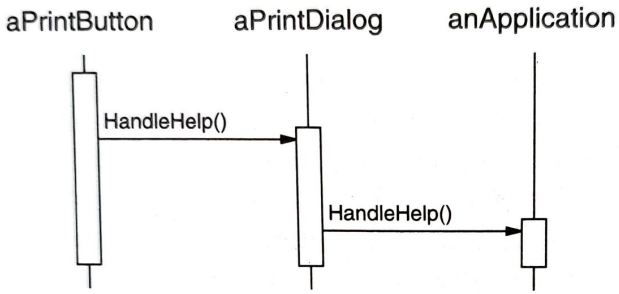
# Idea of Chain of Responsibility

- The DP is applicable in situations where requests may be handled by many handlers.
  - The correct handler is not known beforehands.
- The handlers make a chain. Each hander can:
  - Handle the request itself, or
  - Forward the request to the handler that is next in chain.
- From the client's perspective, this follows the **one-stop shop principle**.
  - The client may stay ignorant of who actually fulfils the request.



# Example 1




- In a GUI, the user may hit the help icon. The help item that is the most specific from the point-of-view of the user context is displayed:
  - Order of decreasing specificity: button → dialog → application

# General structure

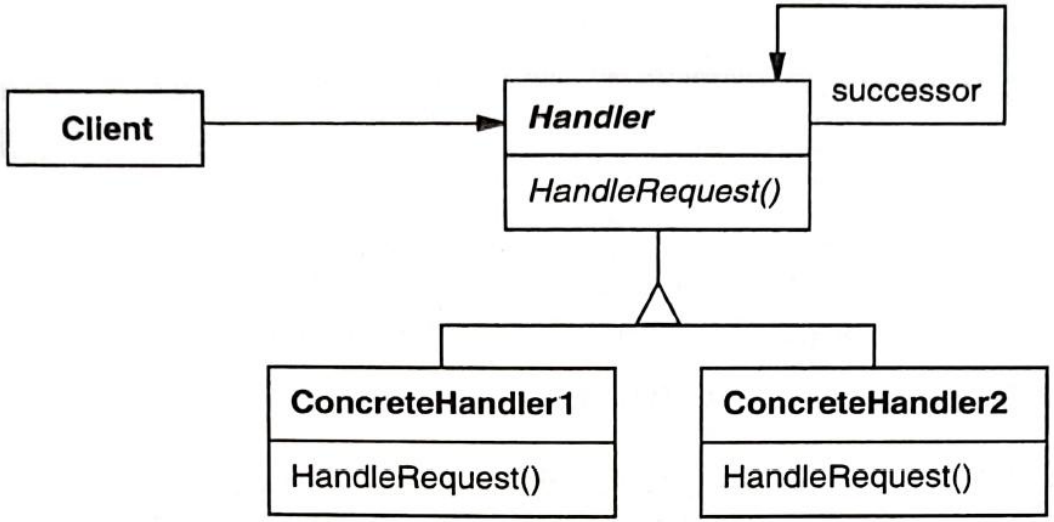



Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 225

# Roles

- **Handler**: declares a common interface for all handlers.
- **Concrete Handler:** Implements the handling method declared in the Handler interface.
- **Client**: submits the request to the handler that is the first in chain.

# Coding the Handler

```
public class Handler {

   private Handler nextHandler;

   public void handle() {
      if (nextHandler != null) {
         nextHandler.handle();
      }
   }
   public void setNextHandler(Handler handler) {
      this.nextHandler = handler;
   }
   public Approver getHandler() {
      return nextHandler;
   }
}
```

- The Handler superclass provides the default implementation for the handle() method.
  - That is, moving the request to the next in chain.
- Note that the Handler class has no abstract methods, so it can be declared concrete.

# Exclusive vs. collaborative handling

- The handing of request can be done alone or together with other handlers.
- **Exclusive handling:** each handler either fully handles the request, or passes it forward to the next handler.
  - The handling method begin with checks of whether the handler is able to fulfil the request.
- **Collaborative handling:** many handlers can contribute to the handling process. A handler may perform its own tasks, and forward the request to the next handler after doing that.
  - E.g. serving a client in a restaurant: take order, prepare food, serve, bill
  - Unlike in the Template Method DP, the sequence may vary dynamically.
  - The handlers are able to terminate the flow in the chain at any point by not calling the next handler.

# Fixed vs. parametrized handlers

- Each handler method may be fixed for a particular purpose, or there can be a parametrized method.
- **Fixed handler**: Provides a specific method for each request type
  - e.g. **handleOrder()**, or just **handle()** if there is only one request type in use
- **Parametrized handler**: There is a single handler method, but the parameter type is passed as a parameter.

# Practical issues

- Careful configuration of the correct handling sequence is necessary.
- It is possible to define a chain where no handler is willing to handle the request.
  - As a consequence, the request is left unhandled.
- There are various way for declaring the successors in the chain:
  - Building the chain can be at client's responsibility.
    - Increases flexibility but requires more code. Maintains loose coupling between concrete handlers.
  - Alternatively, the successor can be coded in the constructor of the concrete handler class.
    - May simplify code, if the handling order is static. On the other hand, makes concrete handlers tightly coupled.