# CMPE 202

## Gang of Four Design Patterns

# Chain of Responsibility

# Motivation

- Sender of a request does not know which object is the right one responsible for handling the request

- Want to decouple the senders and the receivers of a message (i.e. request)

- Would like to give multiple objects a chance to handle the request

# Pros and Cons

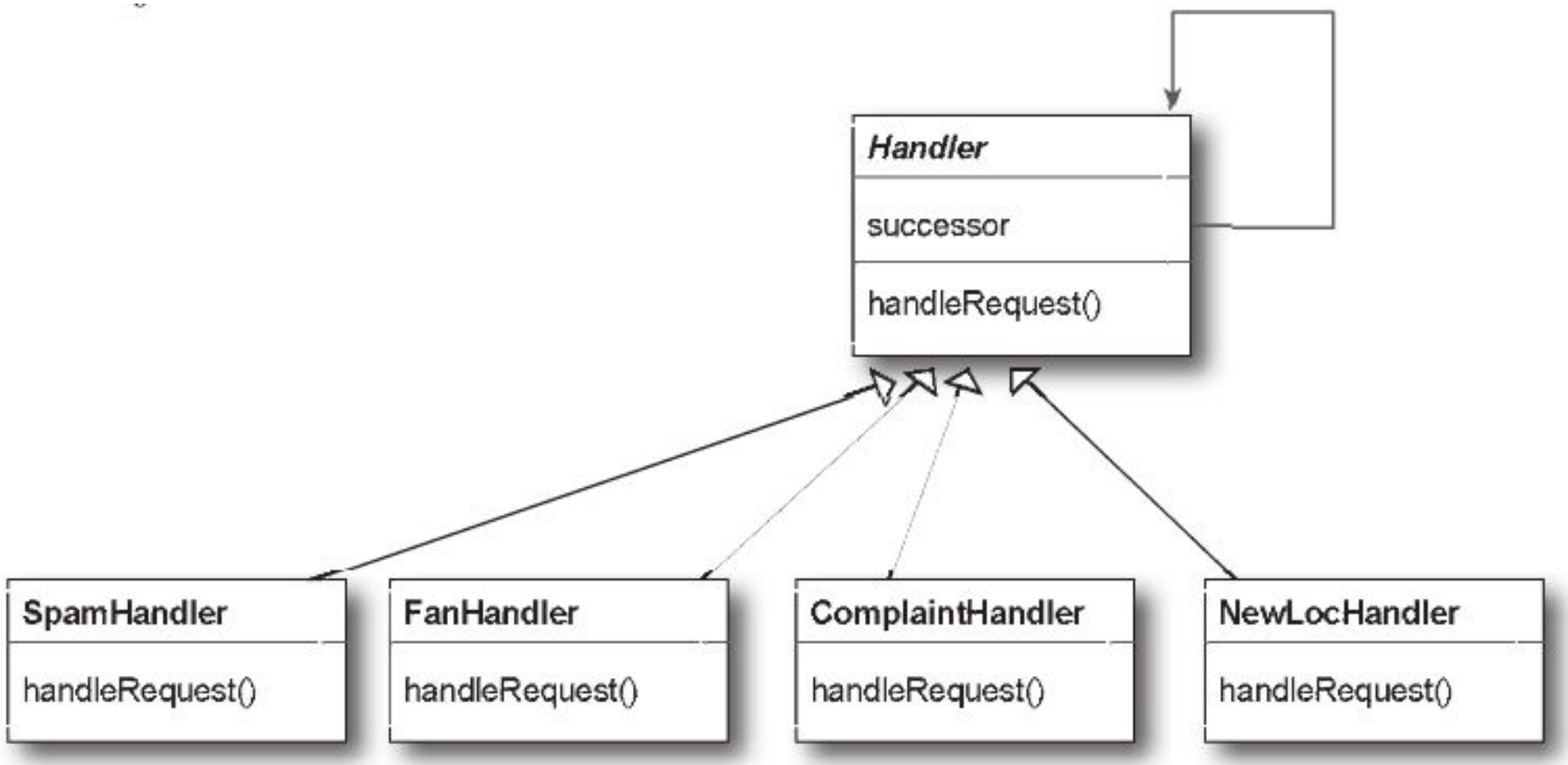
**Chain of Responsibility Benefits**

- Decouples the sender of the request and its receivers.
- Simplifies your object because it doesn't have to know the chain's structure and keep direct references to its members.
- Allows you to add or remove responsibilities dynamically by changing the members or order of the chain.

**Chain of Responsibility Uses and Drawbacks**

- Commonly used in windows systems to handle events like mouse clicks and keyboard events.
- Execution of the request isn't guaranteed; it may fall off the end of the chain if no object handles it (this can be an advantage or a disadvantage).
- Can be hard to observe and debug at runtime.

Each object in the chain acts as a handler and has a successor object. If it can handle the request, it does; otherwise, it forwards the request to its successor.




As email is received, it is passed to the first handler: the SpamHandler. If the SpamHandler can t handle the request, it is passed on to the FanHandler. And so on...

Each email is passed to the first handler.

Email is not handled if it falls off the end of the chain – although, you can always implement a catch-all handler.

# Intent

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

# Applicability

Use Chain of Responsibility when

- more than one object may handle a request, and the handler isn't known *a priori*. The handler should be ascertained automatically.
- you want to issue a request to one of several objects without specifying the receiver explicitly.
- the set of objects that can handle a request should be specified dynamically.

# Participants

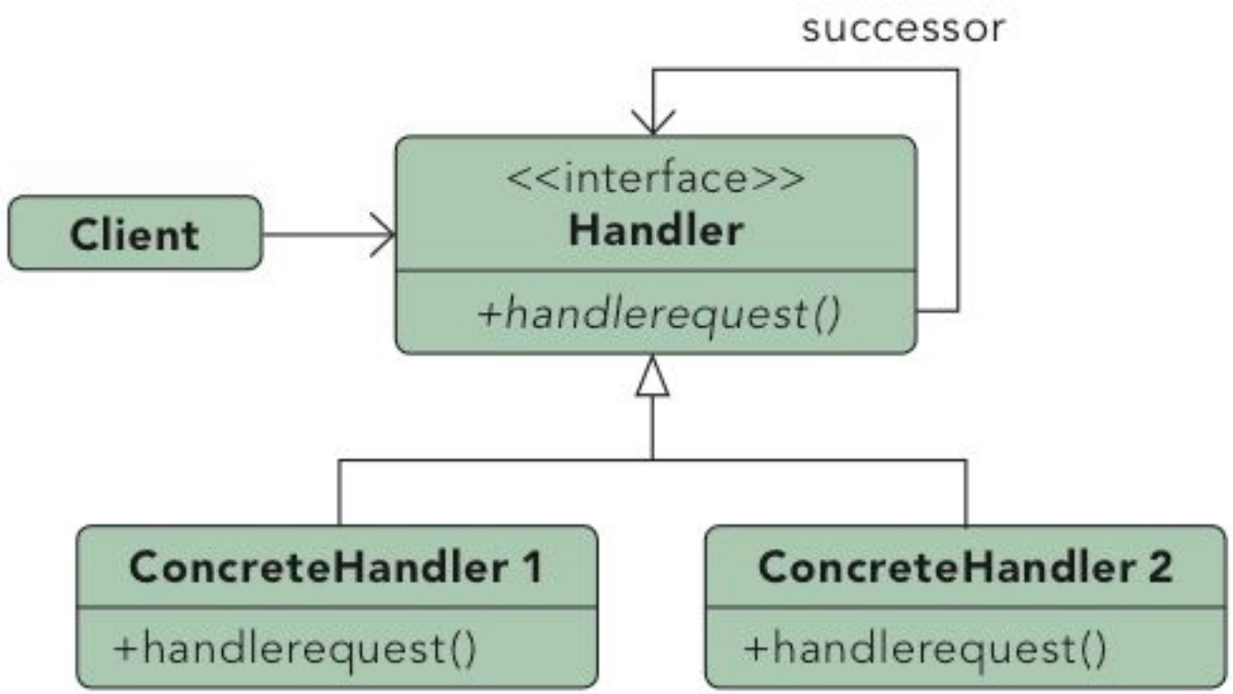
- **Handler** (Interface)
  - defines an interface for handling requests.
  - (optional) implements the successor link.
- **ConcreteHandler**
  - handles requests it is responsible for.
  - can access its successor.
  - if the ConcreteHandler can handle the request, it does so; otherwise it forwards the request to its successor.
- **Client**
  - initiates the request to a ConcreteHandler object on the chain.

# Collaborations

- When a client issues a request, the request propagates along the chain until a ConcreteHandler object takes responsibility for handling it.

# CHAIN OF RESPONSIBILITY

Object Behavioral

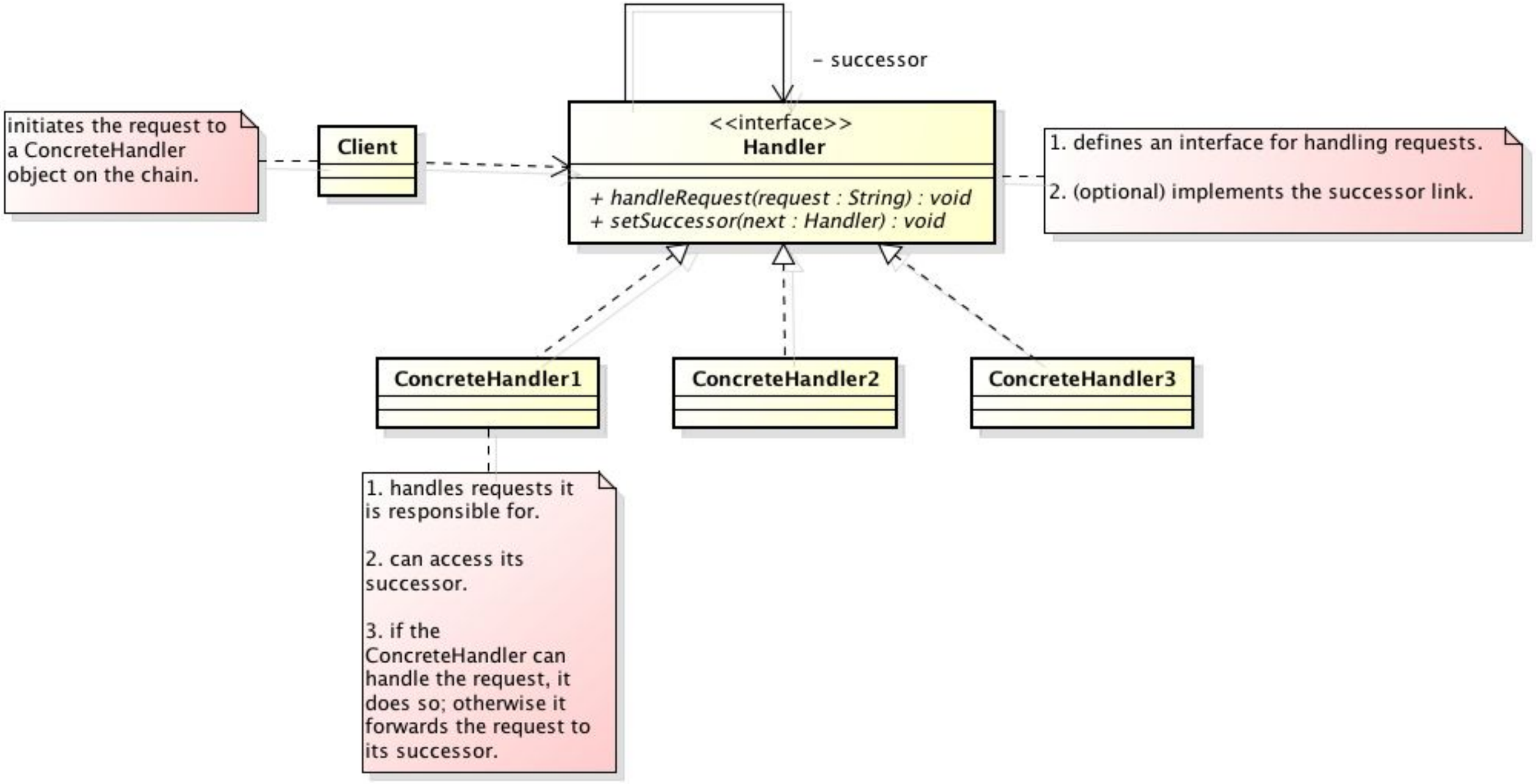


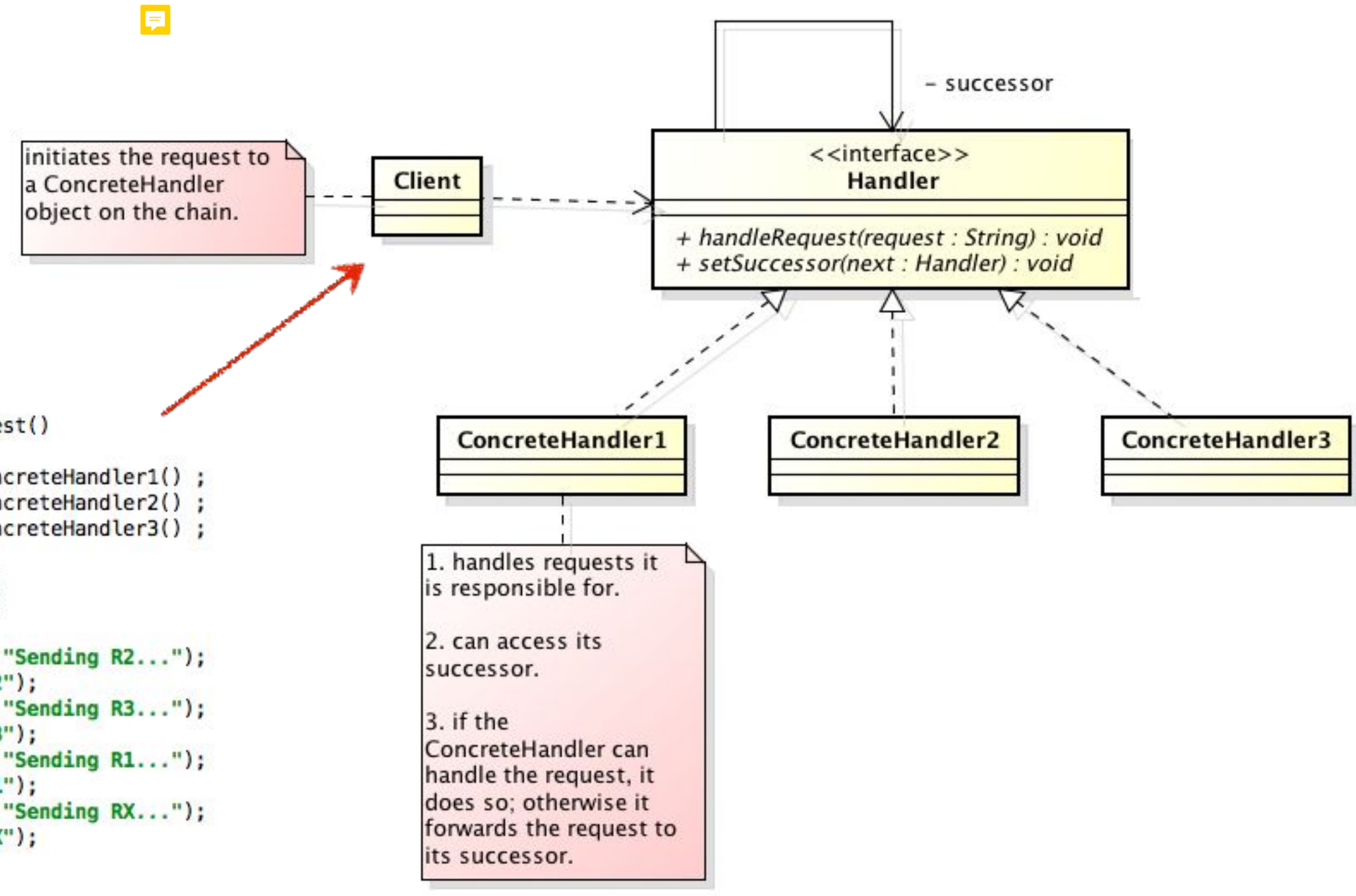

## Purpose

Gives more than one object an opportunity to handle a request by linking receiving objects together.

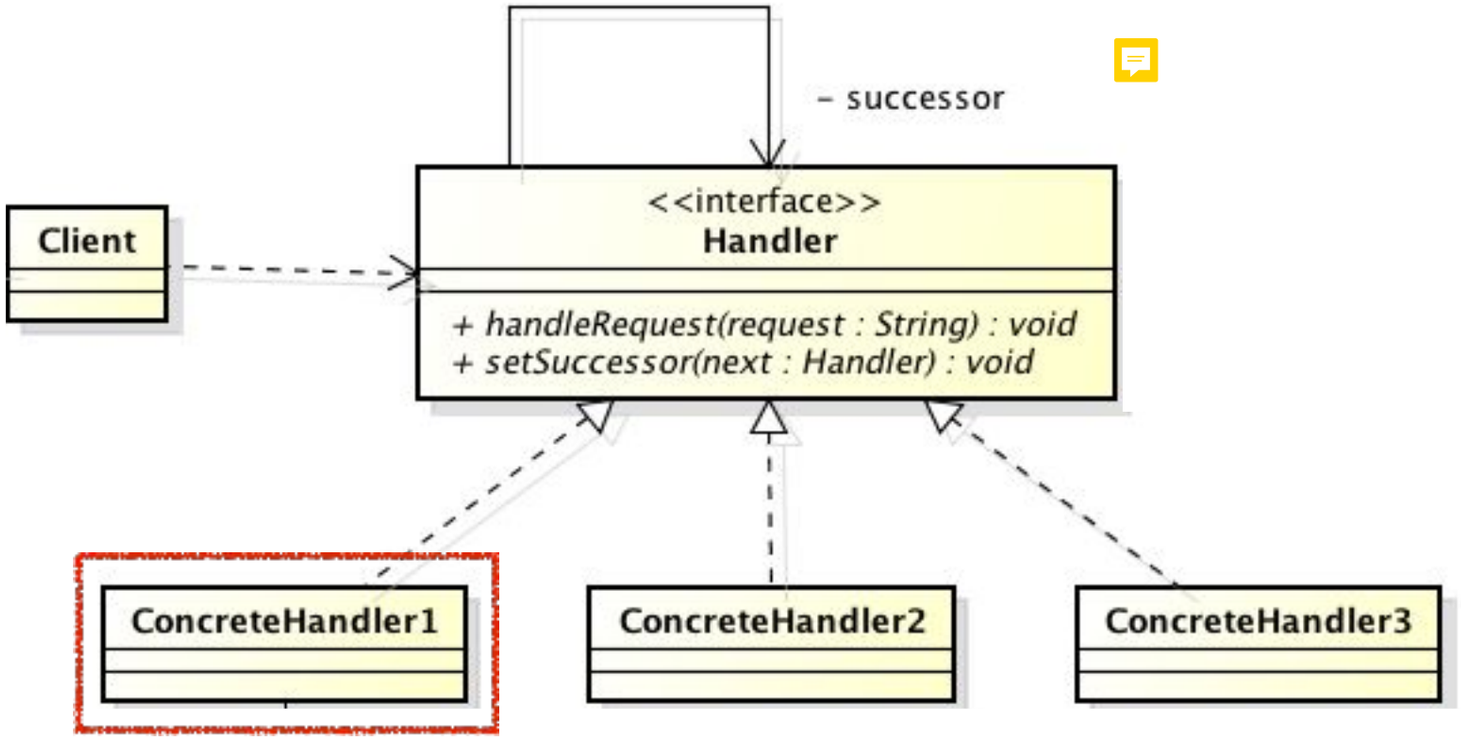
## Use When

- Multiple objects may handle a request and the handler doesn't have to be a specific object.
- A set of objects should be able to handle a request with the handler determined at runtime.
- A request not being handled is an acceptable potential outcome.

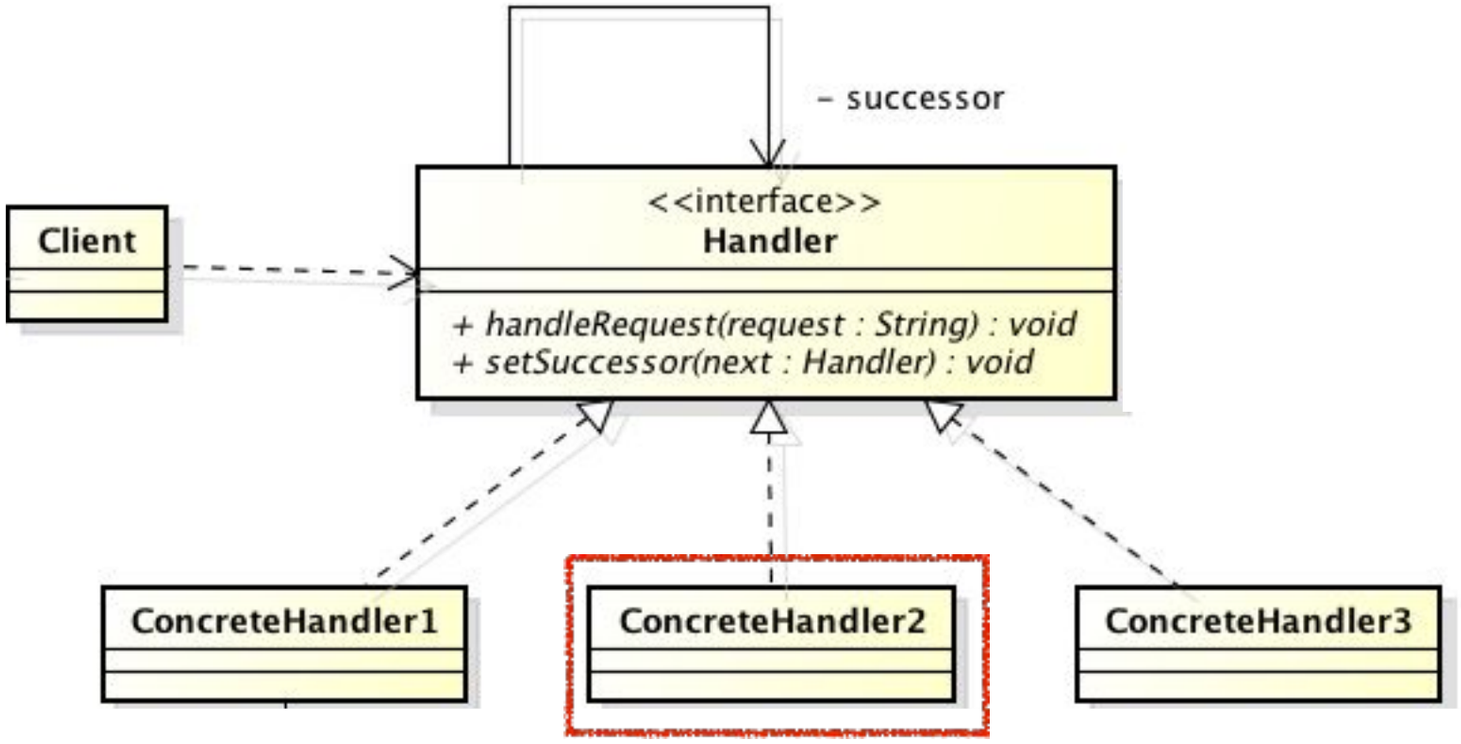

- successor
initiates the request to a ConcreteHandler object on the chain.
Client
<<interface>>
Handler
+ handleRequest(request : String) : void
+ setSuccessor(next : Handler) : void
1. defines an interface for handling requests.
2. (optional) implements the successor link.
ConcreteHandler1
ConcreteHandler2
ConcreteHandler3
1. handles requests it is responsible for.
2. can access its successor.
3. if the ConcreteHandler can handle the request, it does so; otherwise it forwards the request to its successor.



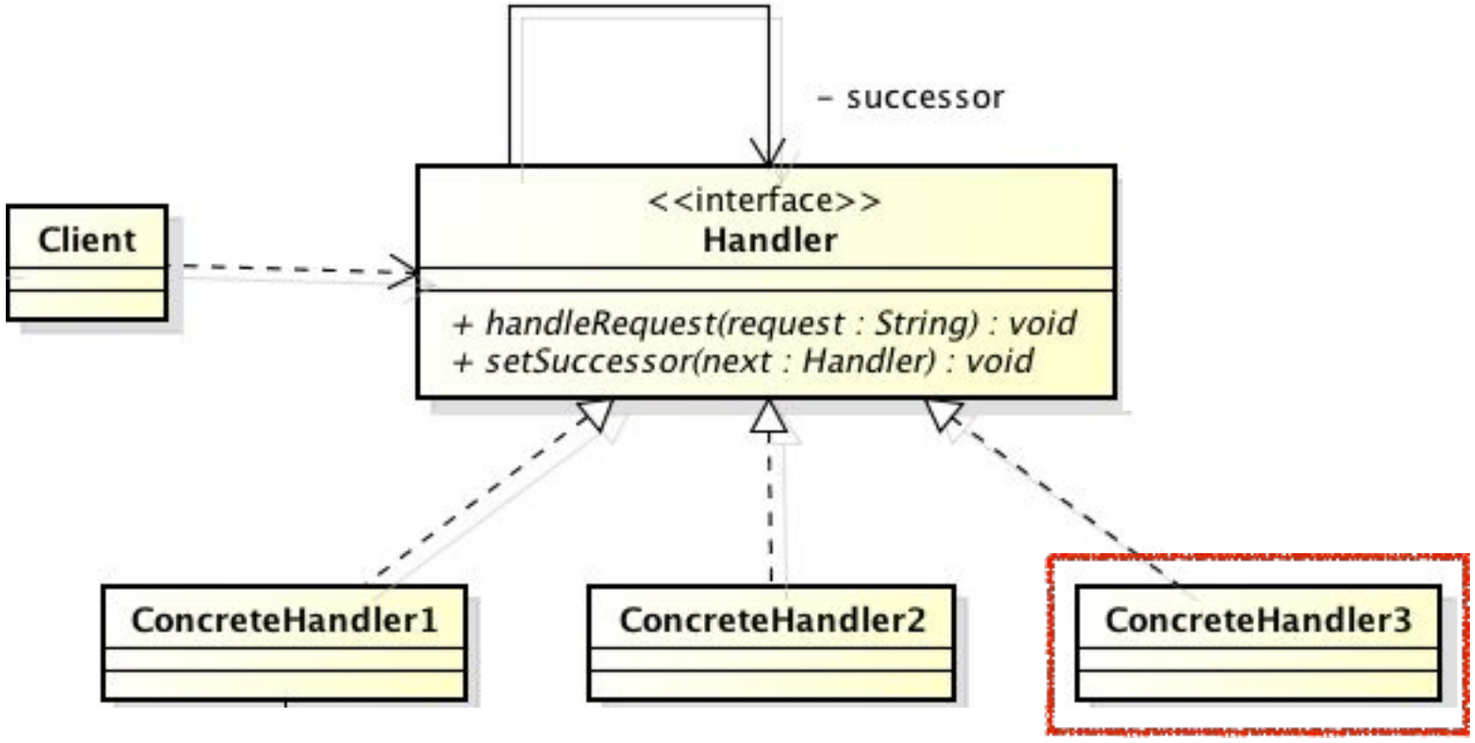

```
public class Client {

    public static void runTest()
    {
        Handler h1 = new ConcreteHandler1() ;
        Handler h2 = new ConcreteHandler2() ;
        Handler h3 = new ConcreteHandler3() ;

        h1.setSuccessor(h2);
        h2.setSuccessor(h3);

        System.out.println( "Sending R2...");
        h1.handleRequest("R2");
        System.out.println( "Sending R3...");
        h1.handleRequest("R3");
        System.out.println( "Sending R1...");
        h1.handleRequest("R1");
        System.out.println( "Sending RX...");
        h1.handleRequest("RX");

    }

}
```



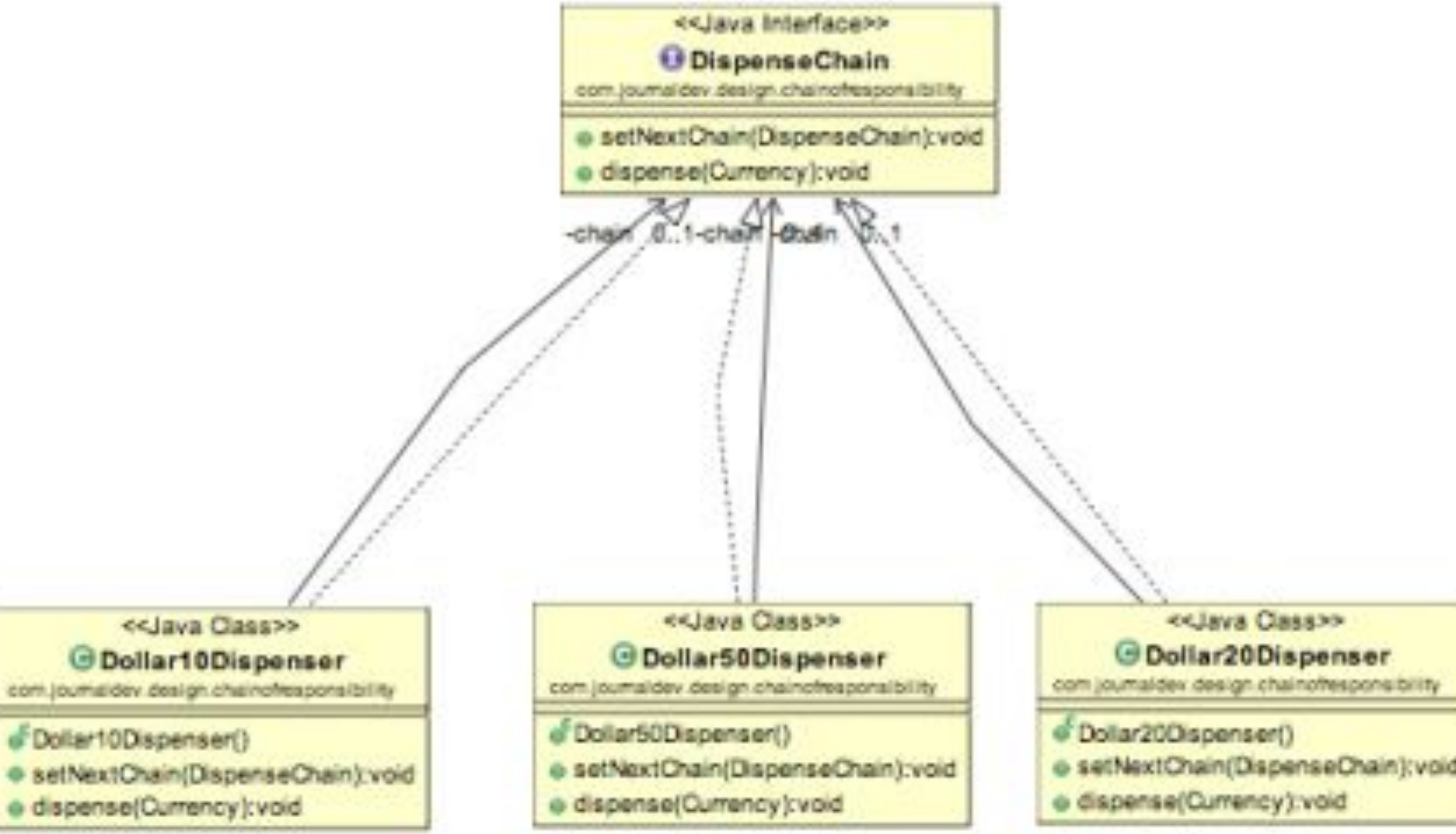

```java
public class ConcreteHandler1 implements Handler {
    private Handler successor = null;

    public void handleRequest( String request ) {
        System.out.println( "R1 got the request...");
        if ( request.equalsIgnoreCase("R1") )
        {
            System.out.println( this.getClass().getName() + " => This one is mine!");
        }
        else
        {
            if ( successor != null )
                successor.handleRequest(request);
        }

    }

    public void setSuccessor(Handler next) {
        this.successor = next ;
    }

}
```




```java
public class ConcreteHandler2 implements Handler {

    private Handler successor = null;

    public void handleRequest( String request ) {
        System.out.println( "R2 got the request...");
        if ( request.equalsIgnoreCase("R2") )
        {
            System.out.println( this.getClass().getName() + " => This one is mine!");
        }
        else
        {
            if ( successor != null )
                successor.handleRequest(request);
        }

    }

    public void setSuccessor(Handler next) {
        this.successor = next ;
    }


}
```




```java
public class ConcreteHandler3 implements Handler {

    private Handler successor = null;

    public void handleRequest( String request ) {
        System.out.println( "R3 got the request...");
        if ( request.equalsIgnoreCase("R3") )
        {
            System.out.println( this.getClass().getName() + " => This one is mine!");
        }
        else
        {
            if ( successor != null )
                successor.handleRequest(request);
        }

    }

    public void setSuccessor(Handler next) {
        this.successor = next ;
    }


}
```


<<Java Interface>>
DispenseChain
com.journaldev.design.chainofresponsibility
setNextChain(DispenseChain):void
dispense(Currency):void
-chain 0..1
-chain
-chain 0..1
<<Java Class>>
Dollar10Dispenser
com.journaldev.design.chainofresponsibility
Dollar10Dispenser()
setNextChain(DispenseChain):void
dispense(Currency):void
<<Java Class>>
Dollar50Dispenser
com.journaldev.design.chainofresponsibility
Dollar50Dispenser()
setNextChain(DispenseChain):void
dispense(Currency):void
<<Java Class>>
Dollar20Dispenser
com.journaldev.design.chainofresponsibility
Dollar20Dispenser()
setNextChain(DispenseChain):void
dispense(Currency):void