Chain of Responsibility Pattern
책임 떠넘기기

01. Chain of Resposibility 패턴

- "책임 떠넘기기"가 필요한 경우가 있다
  - 예: 어떤 요구가 발생했을 때 그 요구를 처리할 객체를 바로 결정할 수 없는 경우에는 다수의 객체를 사슬처럼 연결해 두고 객체의 사슬을 차례로 돌아다니면서 목적에 맞는 객체를 결정하는 경우

- Chain of Responsibility
  - 어떤 사람에게 요구가 들어온다 => 그 사람이 그것을 처리할 수 있으면 처리하고, 처리할 수 없으면 '다음 사람'에게 넘긴다.=> ... ...

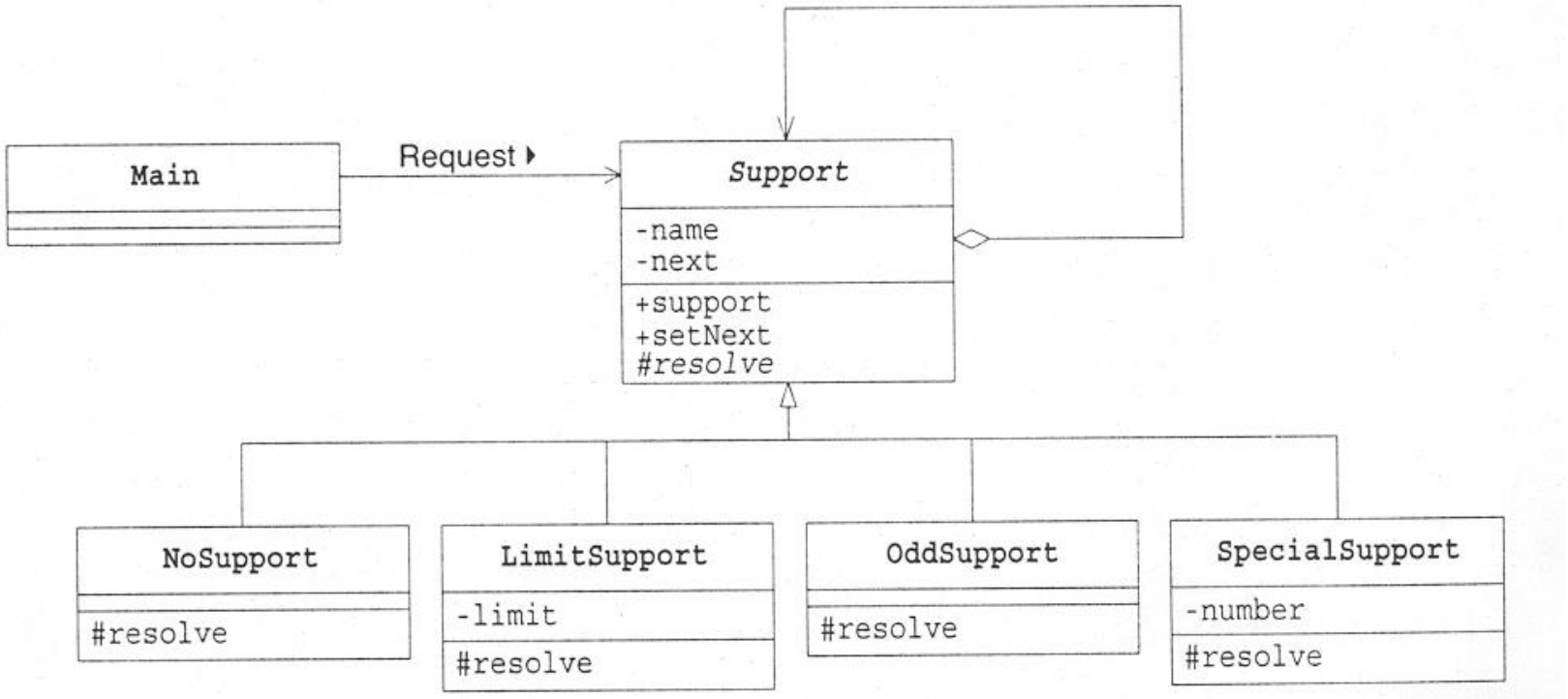
02. 예제 프로그램

- trouble이 발생해서, 누군가가 처리해야 하는 상황

이름	해설
Trouble	발생한 트러블을 나타내는 클래스. 트러블 번호(number)를 갖습니다.
Support	트러블을 해결하는 추상 클래스
NoSupport	트러블을 해결하는 구상 클래스(항상 '처리하지 않습니다')
LimitSupport	트러블을 해결하는 구상 클래스(지정한 번호 미만의 트러블을 해결)
OddSupport	트러블을 해결하는 구상 클래스(홀수 번호의 트러블을 해결)
SpecialSupport	트러블을 해결하는 구상 클래스(지정 번호의 트러블을 해결)
Main	Support들의 연쇄를 만들어 트러블을 일으키는 동작 테스트용 클래스

02. 예제 프로그램

- 클래스 다이어그램



Trouble 처리자들

02. 예제 프로그램

- Trouble 클래스
  - number 필드: 트러블 번호를 유지함
  - getNumber( )
    - 트러블 번호를 반환하는 메소드

02. 예제 프로그램

- Support 클래스
  - 트러블을 해결할 연쇄를 만들기 위한 추상 클래스
  - name 필드: 트러블 해결자의 이름
  - next 필드:
    - 자기 다음 번에 트러블을 해결할 객체(떠넘기기를 할 곳)를 가리킨다.
  - setNext( )
    - next 필드를 설정한다
  - suuport( )
    - resolve( ) 메소드를 호출해서, 반환 값이 false 이면, 자기 다음 해결자에게 트러블을 떠넘긴다.
    - Template Method 패턴을 사용했다.

02. 예제 프로그램

- NoSupport 클래스
  - Support의 하위 클래스
  - resolve( )
    - 항상 false를 반환한다.
    - 즉, 어떠한 트러블로 해결하지 않는다.

- LimitSupport 클래스
  - Support의 하위 클래스
  - limit로 지정한 번호 미만의 트러블 만을 해결하는 클래스

02. 예제 프로그램

- OddSupport 클래스
  - 홀수 번호의 트러블을 처리하는 클래스

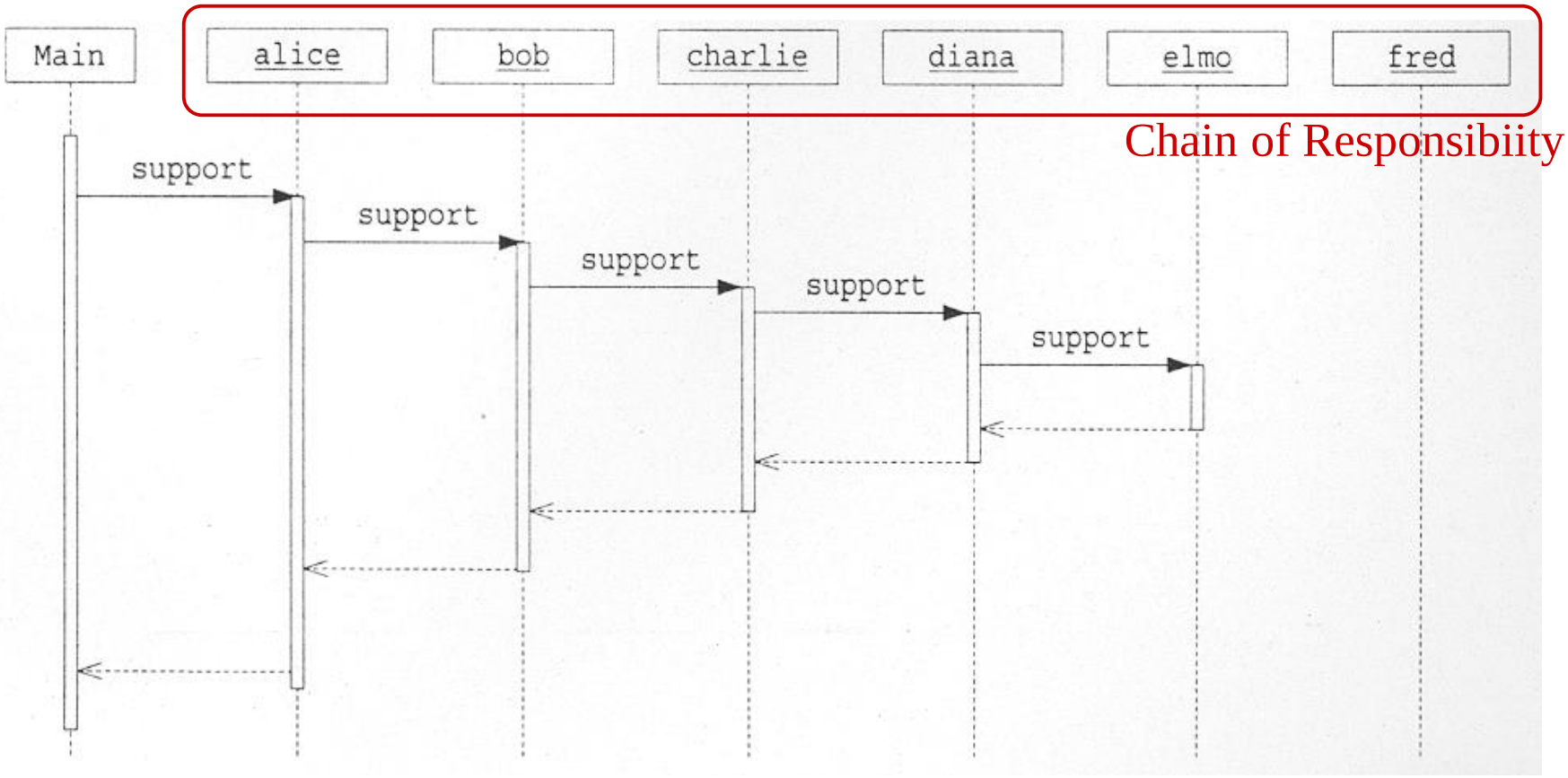
- SpecialSupport 클래스
  - 지정한 번호의 트러블에 한해서 처리하는 클래스

02. 예제 프로그램

- Main 클래스
  - 알고리즘
    - 여섯 명의 트러블 해결자 생성
    - Chain of Responsibility 형성
    - 다양한 트러블 발생시켜서, Alice에게 전달해 준다.
  - 결과
    - 처음에는 Bob이 분발한다
    - Bob이 해결할 수 없을 때 Diana가 등장한다.
    - Alice는 전혀 등장하지 않는다 <= 모든 트러블을 떠넘기므로
    - 429번 트러블은, Charlie가 해결한다.

02. 예제 프로그램

- Main 클래스
  - 363번 트러블을 처리할 때의 시퀀스 다이어그램

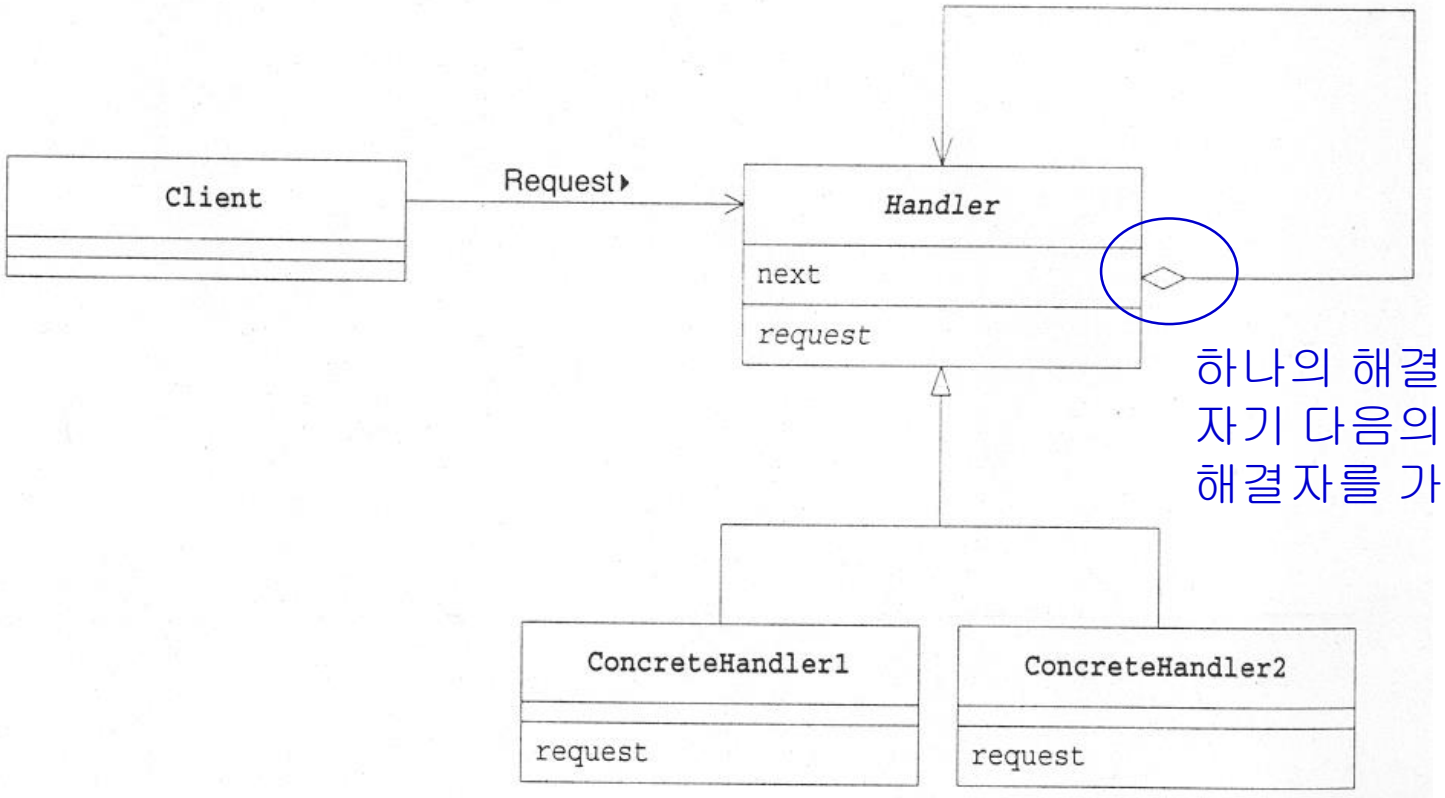


03. 등장 역할

- Handler(처리자)의 역할
  - 요구를 처리하는 클래스들의 인터페이스(API)를 정하는 역할
  - 예제에서는, Support 클래스가 해당됨
    - 요구를 처리하는 메소드는, support( )이다.
- ConcreteHandler(구체적인 처리자)의 역할
  - 요구를 처리하는 구체적인 역할
  - 예제에서는, NoSupport, LimitSupport, OddSupport, SpecialSupport 클래스가 해당됨
- Client(요구자)의 역할
  - ConcreteHandler 역할에게 요구를 하는 역할
  - 예제에서는, Main 클래스가 해당됨

03. 등장 역할

- 클래스 다이어그램



하나의 해결자는,
자기 다음의
해결자를 가지고 있다.

04. 독자의 사고를 넓혀주는 힌트

- 요구하는 사람과 요구를 처리하는 사람을 느슨하게 연결
  - Client는 맨 처음 사람에게만 요구를 한다.
  - 그 요구가 처리자들을 연결한 사슬을 돌아다니다가,
  - 적절한 처리자에 의해 처리된다.
  - 이 패턴을 사용하지 않으면, "이 요구는 이 처리자가 처리해야 한다"라는 지식을 누군가 중앙 집중적으로 가지고 있어야 한다.
    - 이 지식을 요구자에게 맡기는 것은 현명하지 않다.

04. 독자의 사고를 넓혀주는 힌트

- 동적으로 연쇄의 형태를 바꿀 수 있다
  - Alice ~ Fred 까지의 서포트 팀이 항상 고정되는 것은 아니다.
  - 동적으로 처리자들의 순서를 바꿀 수 있다.

- 자신의 일에 집중할 수 있다
  - 자신이 할 수 없으면 재빨리 '다음 사람'에게 넘김으로써, 각각의 처리자가 자신의 일에만 집중할 수 있다.

- 떠넘기기로 처리가 늦어지지 않을까?
  - 유연성은 높지만, 처리 속도는 느리다.
  - 요구와 처리자의 관계가 고정적이고, 처리의 속도가 중요한 경우에는 이 패턴을 사용하지 않는 편이 유효할 수도 있다.

05. 관련 패턴

- Composite 패턴
  - Handler 역할에서 Composite 패턴이 등장하는 경우가 자주 있음

- Command 패턴
  - Handler 역할에게 제시된 ' 요구'에 Command 패턴이 사용되는 경우가 있음

06. 요약

- 요구를 처리하는 객체들을 사슬 모양으로 늘어놓고, 요청이 들어오면 누가 처리할 지를 차례차례 체크해 나가는 Chain of Responsibility 패턴