Visitor Pattern
구조 안을 돌아다니면서 일을 한다.

01. Visitor 패턴

- 데이터 구조 안에 저장되어 있는 많은 요소에 대해서, 무언가 "처리"해 나가고자 한다.

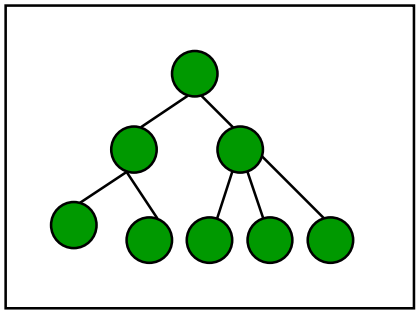
  - 얼핏 생각하면, 데이터 구조를 나타내고 있는 클래스 안에 "처리"를 기술해야 한다고 생각할 것이다.

  - 그러나, "처리"가 여러 종류라고 한다면?
    - 새로운 처리가 필요해질 때마다, 데이터 구조를 나타내는 클래스를 수정해야 한다. => 다른 방법이 필요하다.

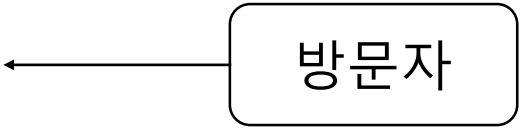
01. Visitor 패턴

- 데이터 구조와 처리를 분리하자.
  - 데이터 구조를 돌아다니는 "방문자"를 정의해서, 이 방문자가 "처리"를 담당하도록 하자.
  - 새로운 처리를 추가하고 싶을 때는, 새로운 "방문자"를 만든다.

- 데이터 구조는, 문을 두드리는 "방문자"를 받아들이기만 하면 된다.



데이터 구조

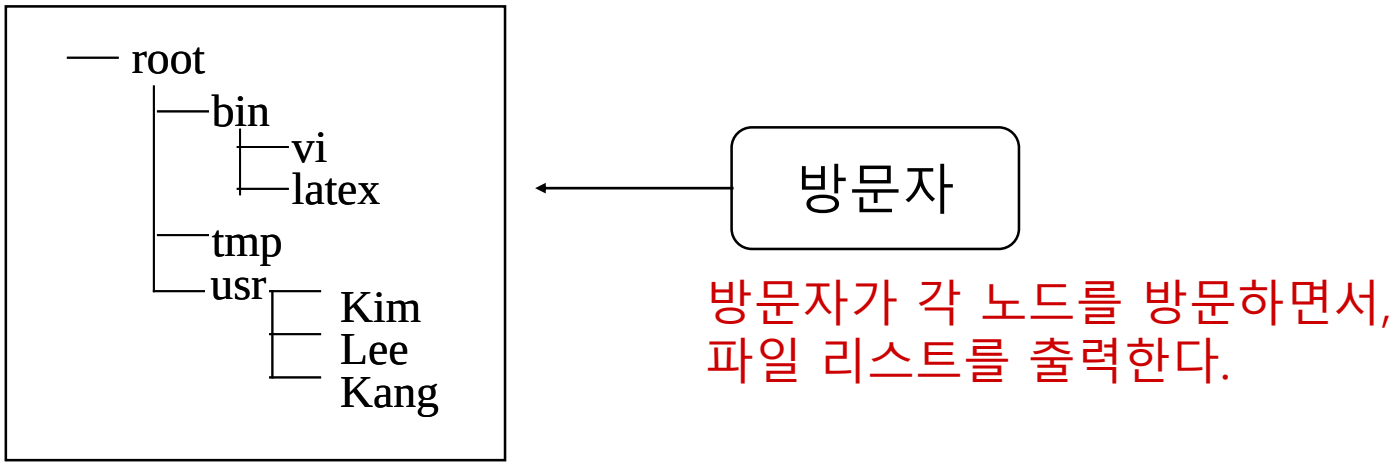


방문자가 데이터 처리를 담당한다.

02. 예제 프로그램

- 방문자가 방문하는 데이터 구조
  - 11장 Composite 패턴의 예제에서 사용된 파일과 디렉토리를 다시 사용한다.

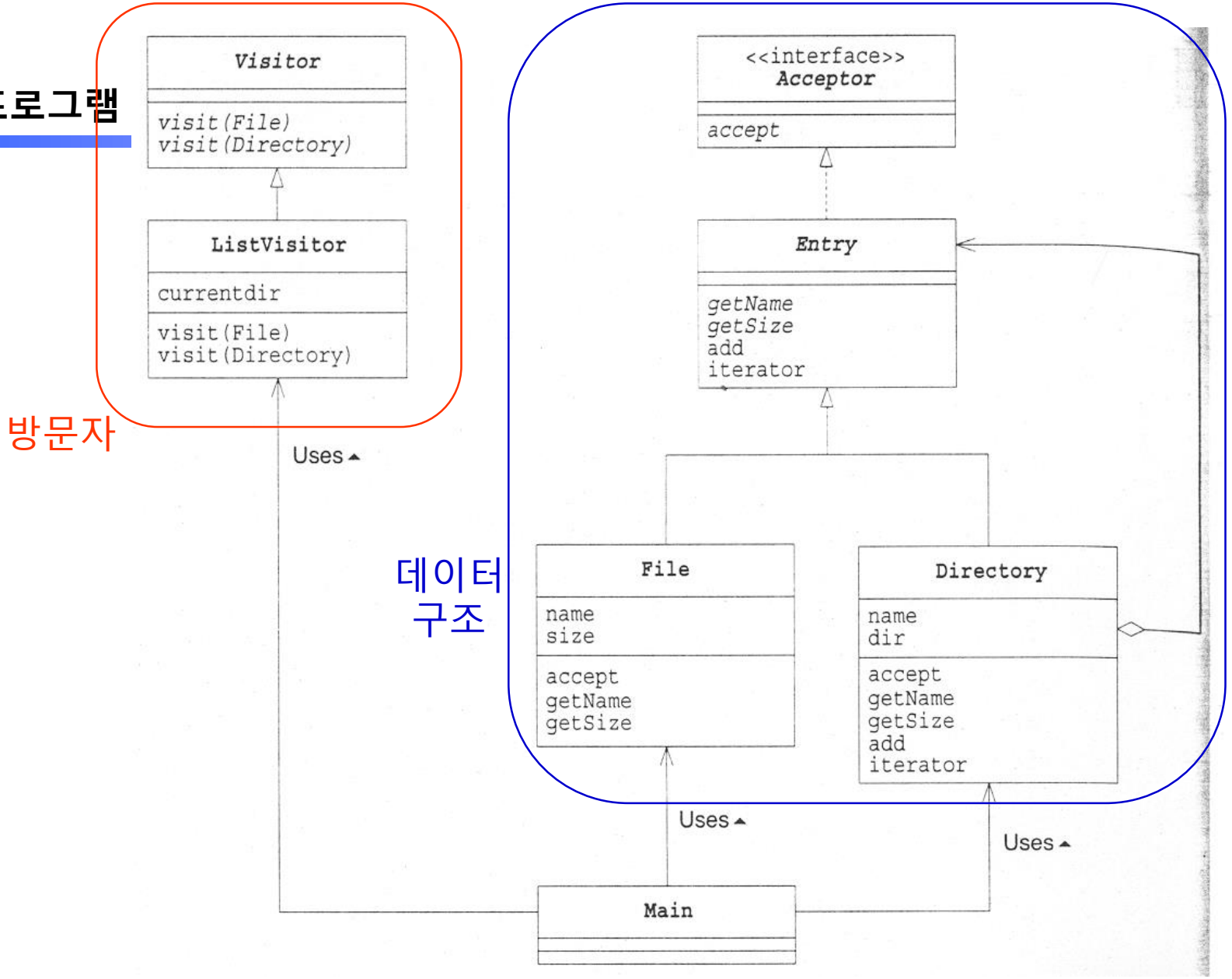
- 파일과 디렉토리로 구성된 데이터 구조를 방문자가 방문하면서, 파일 리스트를 출력한다.



02. 예제 프로그램

이름	해설
Visitor	파일이나 디렉토리를 방문하는 방문자를 나타낸 추상 클래스
Acceptor	Visitor 클래스의 인스턴스를 받아들이는 데이터 구조를 나타내는 인터페이스
ListVisitor	Visitor 클래스의 하위 클래스로 파일이나 디렉토리의 일람을 나타내는 클래스
Entry	File과 Directory의 상위 클래스가 되는 추상 클래스(Acceptor 인터페이스를 구현)
File	파일을 나타내는 클래스
Directory	디렉토리를 나타내는 클래스
FileTreatmentException	File에 대해서 add한 경우에 발생하는 예외 클래스
Main	동작 테스트용 클래스

02. 예제 프로그램


Visitor
visit(File)
visit(Directory)
ListVisitor
currentdir
visit(File)
visit(Directory)
방문자
Uses
<<interface>>
Acceptor
accept
Entry
getName
getSize
add
iterator
데이터
구조
File
name
size
accept
getName
getSize
Directory
name
dir
accept
getName
getSize
add
iterator
Uses
Uses
Main

02. 예제 프로그램

- Visitor 클래스
  - 방문자를 나타내는 추상 클래스
  - visit(File)
    - File을 방문했을 때 File 클래스가 호출하는 메소드
  - visit(Directory)
    - Directory를 방문했을 때 Directory 클래스가 호출하는 메소드
  - 메소드 오버로드(overload) 이용했음
    - visit 메소드의 이름은 같은데, 인자의 종류가 다르다.

02. 예제 프로그램

- Acceptor 인터페이스
  - 방문자를 받아들이는 클래스를 위한 인터페이스
  - accept(Visitor v)
    - Visitor 타입을 입력 인자로 받아들임

02. 예제 프로그램

- Entry 클래스
  - Composite 패턴에 등장했던 File이나 Directory를 위한 추상 클래스
  - Acceptor 인터페이스를 구현함
  - add( )
    - Directory 클래스에서만 add( )가 유효하므로, Entry 클래스에서는 일단 에러로 처리한다.
  - iterator( )
    - 요소에 대한 Iterator를 얻을 때 호출되는 메소드
    - Directory 클래스에서만 iterator( )가 유효하므로, Entry 클래스에서는 일단 에러로 처리한다.

02. 예제 프로그램

- File 클래스
  - accept(Visitor)
    - 클라이언트가, File 객체에게 "방문자를 받아들이세요"라고 요청할 때 호출하는 메소드
      - 클라이언트는 Visitor를 매개변수로 하여 accept를 호출할 것이다.
    - 입력 인자로 들어온 방문자의 visit 메소드를 호출한다.
      - 이 때, 현재 자신 객체를 인자로 하여 호출한다.
      - 그러면, Visitor의 visit(File) 메소드가 실행된다.
      - 즉, 방문자가 방문하면 방문자에게 "나는 File 객체입니다. 나의 일을 처리해 주세요"라고 요청하는 것과 비슷하다.

02. 예제 프로그램

- Directory 클래스
  - iterator( )
    - 디렉토리가 유지하고 있는 엔트리들에 대한 Iterator를 반환한다.
  - accept(Visitor)
    - 입력 인자로 들어온 Visitor에게, 자기 자신을 매개 변수로 해서 visit( )을 호출한다.
    - 그러면, Visitor의 visit(Directory) 메소드가 실행된다.
      - 방문자가 방문하면 방문자에게 "나는 Directory 객체입니다. 나의 일을 처리해 주세요"라고 요청하는 것과 비슷하다.

02. 예제 프로그램

- ListVisitor 클래스
  - Visitor 클래스의 하위 클래스
  - 실제 데이터구조(File이나 Directory)를 옮겨 다니면서, 리스트를 출력하는 일을 한다.
  - currentdir 필드
    - 현재 주목하고 있는 디렉토리 명을 저장함
  - visit(File)
    - 입력인자로 받아들인 "File에 대해 수행해야 할 처리"가 기술되어 있다.
    - 알고리즘
      - 현재 디렉토리와 File의 toString( ) 반환 값을 연결하여
      - 현재 파일의 전체 경로를 출력한다.

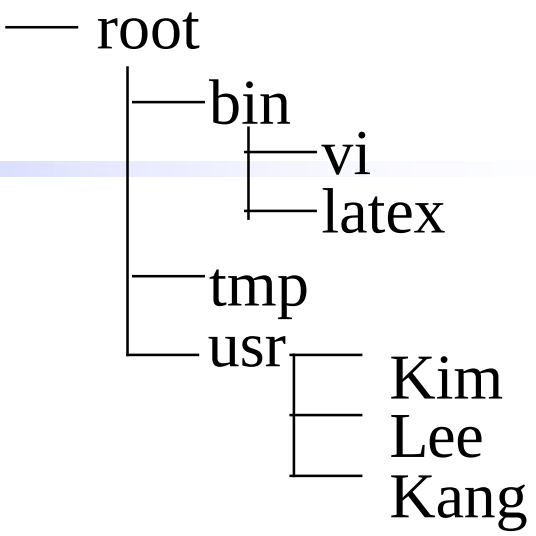
02. 예제 프로그램

- ListVisitor 클래스
  - visit(Directory)
    - 입력 인자로 받아들인 "Directory에 대해 수행해야 할 처리"가 기술되어 있다.
    - 알고리즘
      - 먼저, 디렉토리 전체 경로 및 크기를 출력한다.
      - 현재 디렉토리(currentdir)를 임시로 savedir에 저장한다
      - 현재 디렉토리(currentdir)를, 입력 인자로 들어온 디렉토리로 바꾼다.
      - 입력 인자로 들어온 디렉토리의 iterator를 얻는다.
      - 입력 인자로 들어온 디렉토리가 유지하는 원소들을 차례로 방문하면서, accept(this)를 호출하여 방문자가 방문했음을 알린다.
      - while 루프가 끝나면, currentdir을 원래 디렉토리로 복귀시킨다.
    - 복잡한 재귀적인 호출
      - accept 메소드는 visit 메소드를 호출하고, visit 메소드는 accept 메소드를 호출한다.

02. 예제 프로그램

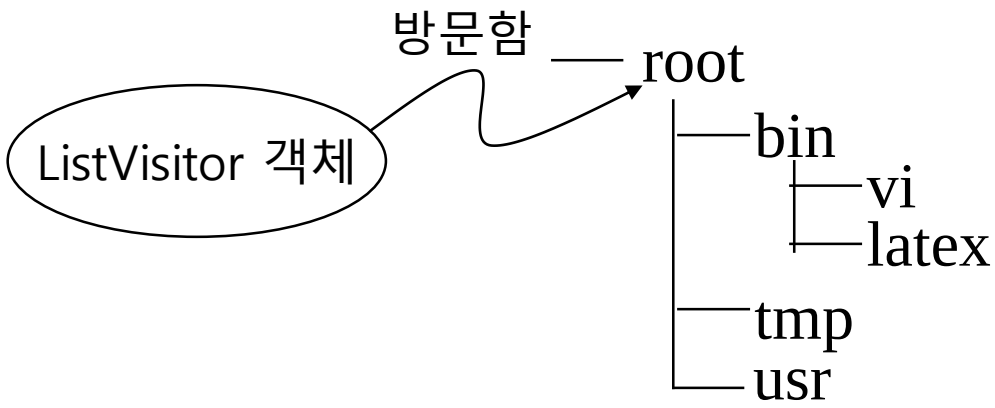
- FileTreatmentException 클래스
  - File 엔트리에 무엇인가 추가(add)하고자 할 때 발생되는 예외

02. 예제 프로그램



- Main 클래스
  - Composite 패턴에서는,
    - main( ) 메소드가 엔트리 리스트를 출력하기 위해서 rootdir.printList( )를 호출하였다.
    - 엔트리의 내용을 출력하는 책임을, File이나 Directory 클래스가 가지고 있다.
  - Visitor 패턴에서는,
    - main( ) 메소드가 엔트리 리스트를 출력하기 위해서, rootdir.accept(new ListVisitor( ))를 호출하였다.
    - 엔트리의 내용을 출력하는 책임을, ListVisitor 클래스가 가지고 있다.

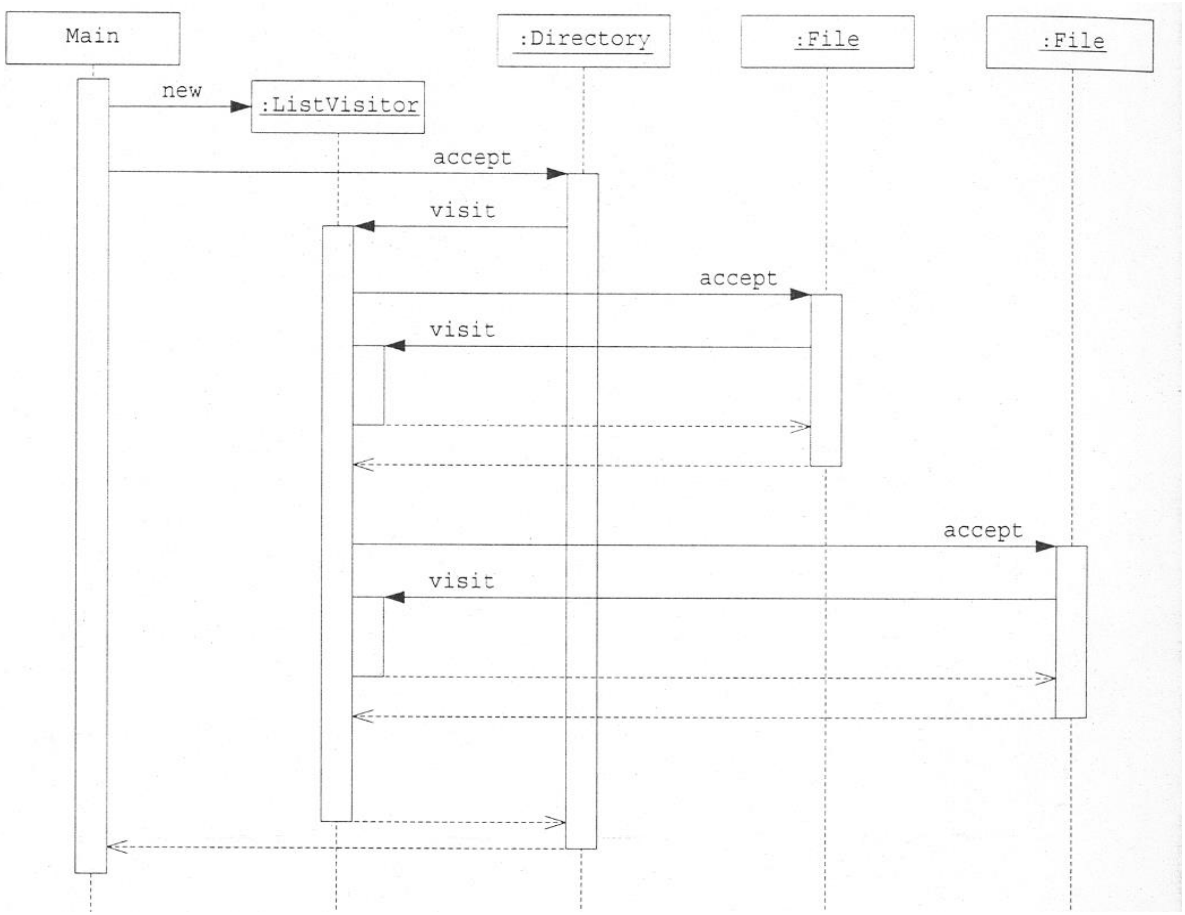
02. 예제 프로그램



i) 방문한 곳이 디렉토리이면,
현재 디렉토리 경로 및 크기를 출력하고,
각 내용물에 대해서, 방문자를 받아들이라고 요청한다.
ii) 방문한 곳이 파일이면,
현재 디렉토리와 파일 크기를 출력한다.

02. 예제 프로그램

- Visitor와 Acceptor의 상호 호출
  - 하나의 Directory와 두 개의 File이 있는 경우의 시퀀스 다이어그램



03. 등장 역할

- Visitor(방문자)의 역할
  - 데이터 구조 내의 각각의 구체적인 요소(ConcreteAcceptor 역할)에 visit(xxx) 메소드를 선언하는 역할
  - visit(xxx) 메소드는, 실제 xxx를 처리하기 위한 실제 코드가 하위 클래스에 의해 제공된다.
  - 예제에서는 Visitor 클래스가 해당됨

- ConcreteVisitor(구체적 방문자)의 역할
  - Visitor 역할의 인터페이스(API)를 실제로 구현하는 역할
  - 예제에서는 ListVisitor 클래스가 해당됨

03. 등장 역할

- Acceptor(수용자)의 역할
  - Visitor 역할이 방문할 장소를 나타내는 역할
  - 방문자를 받아들이는 accept(Visitor) 메소드를 선언한다.
  - 예제에서는, Accept 인터페이스가 해당됨

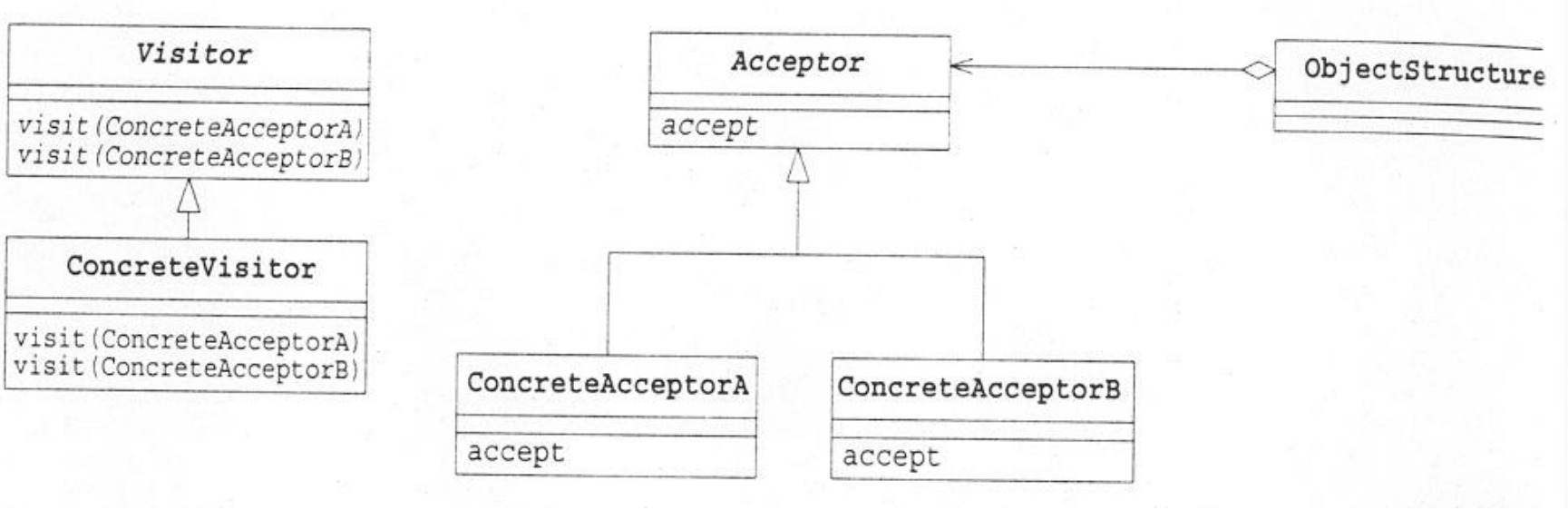
- ConcreteAcceptor(구체적인 수용자)의 역할
  - Acceptor 역할의 인터페이스를 구현하는 역할
  - 예제에서는, File이나 Directory 클래스가 해당됨

03. 등장 역할

- ObjectStructure(객체의 구조)의 역할
  - Acceptor 역할을 집합으로 취급할 수 있도록 해 주는 메소드를 제공한다.
  - 예제에서는, Directory 클래스가 해당됨
    - 유지하고 있는 요소들을 얻어갈 수 있도록, iterator 메소드를 제공함

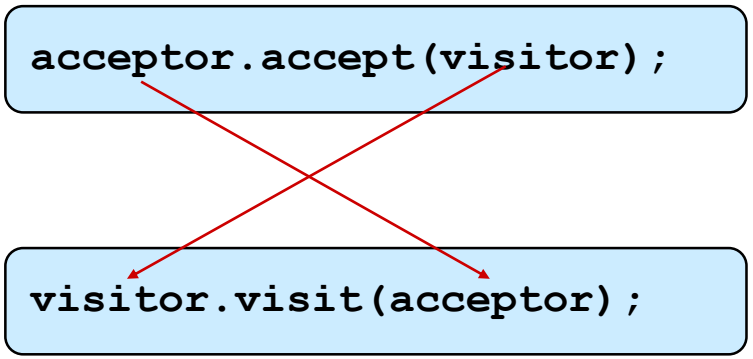
03. 등장 역할

- 클래스 다이어그램



04. 독자의 사고를 넓혀주는 힌트

- 더블 디스패치(dispatch)
  - dispatch: 급파하다, 발송하다, 신속히 처리하다.
  - Visitor와 Acceptor는 서로 대응 관계
    - ConcreteAcceptor와 ConcreteVisitor의 역할을 하는 한 쌍에 의해 실제의 처리가 결정된다.



acceptor에게 방문자를 받아들이라고 요청하면

acceptor는, 자신을 인자로 해서 방문자의 visit을 호출한다.

04. 독자의 사고를 넓혀주는 힌트

- 왜 이렇게 복잡한 일을 하는가?
  - "반복 처리가 필요하면 데이터 구조 안에 루프를 쓰면 되지 않나?"
  - Visitor 패턴의 목적은, "처리"를 "데이터 구조"로부터 분리하는 것이다.
    - 다른 처리를 하는 ConcreteVisitor를 추가할 수 있다. (연습문제 참조)
    - 또, 기존의 ConcreteVisitor의 기능을 확장하기도 쉽다.
    - 결국, 부품의 독립성을 높여준다.

04. 독자의 사고를 넓혀주는 힌트

- Open-Closed Principle
  - 확장에 대해서는 열려있고
    - 클래스를 설계할 때에 특별한 이유가 없는 한 장래의 확장을 허락해야 한다.
  - 수정에 대해서는 닫혀있다.
    - 확장을 하더라도 기존의 클래스는 수정할 필요가 없어야 한다.

- ConcreteVisitor 역할의 추가는 간단하지만, ConcreteAcceptor 역할의 추가는 어렵다.
  - why:
    - Acceptor의 구조가 바뀌면, 이를 처리하는 모든 Visitor의 내부도 바꿔야 하므로

05. 관련 패턴

- Iterator 패턴
  - Iterator 패턴과 Visitor 패턴 모두 어떤 데이터 구조상에서 처리를 실행하는 것임
  - Iterator 패턴은 데이터 구조가 보관하고 있는 요소를 하나 하나 얻는데 사용함
  - Visitor 패턴은 데이터 구조가 보관하고 있는 요소에 특정의 처리를 가하는데 사용함
- Composite 패턴
  - 방문처가 되는 데이터 구조는 Composite 패턴이 되는 경우가 있음

06. 요약

- 데이터 구조 안을 돌아다니면서 처리를 수행하는 Visitor 패턴