디자인 패턴 – 패턴 응용

송태국
넥스트리 소프트㈜ 부사장/CTO
tsong@nextree.co.kr

목차

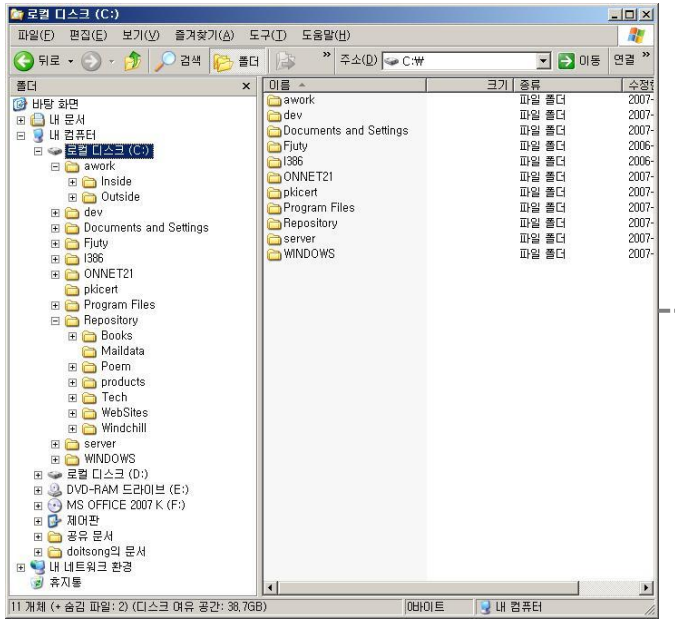
1. 파일시스템 요구사항(1/2)

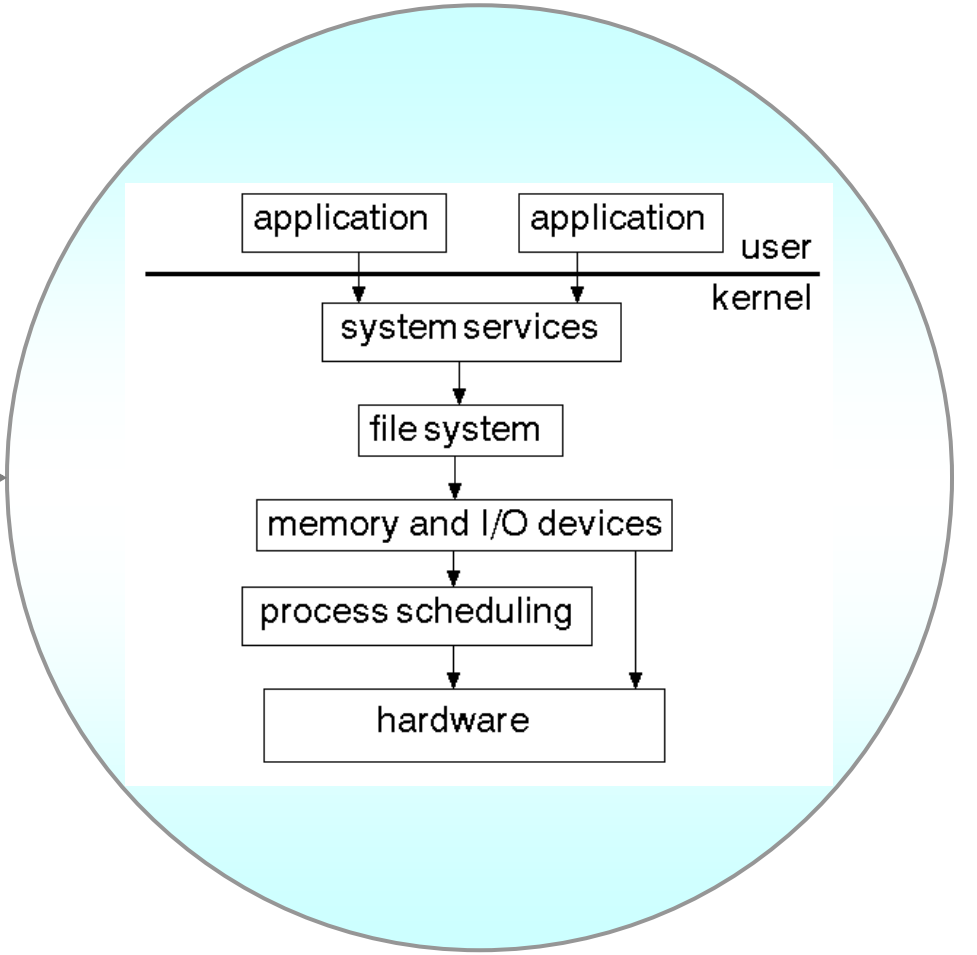
NEXTREE

- ✓ 사용자 관점
  - 임의의 크기와 복잡성을 가진 파일 구조를 처리할 수 있어야 함
  - 파일 구조의 깊이와 넓이에 한계가 없어야 함
- ✓ 클라이언트(explorer) 프로그래머 관점
  - 클라이언트가 파일 구조를 다루기 쉽고 확장하기 쉬워야 함
  - 폴더와 파일을 이름을 이용하여 같은 방식으로 다룰 수 있어야 함
  - 재-구현없이 새로운 유형의 파일(예, symbolic links)을 수용할 수 있어야 함

1. 파일시스템 요구사항(2/2)

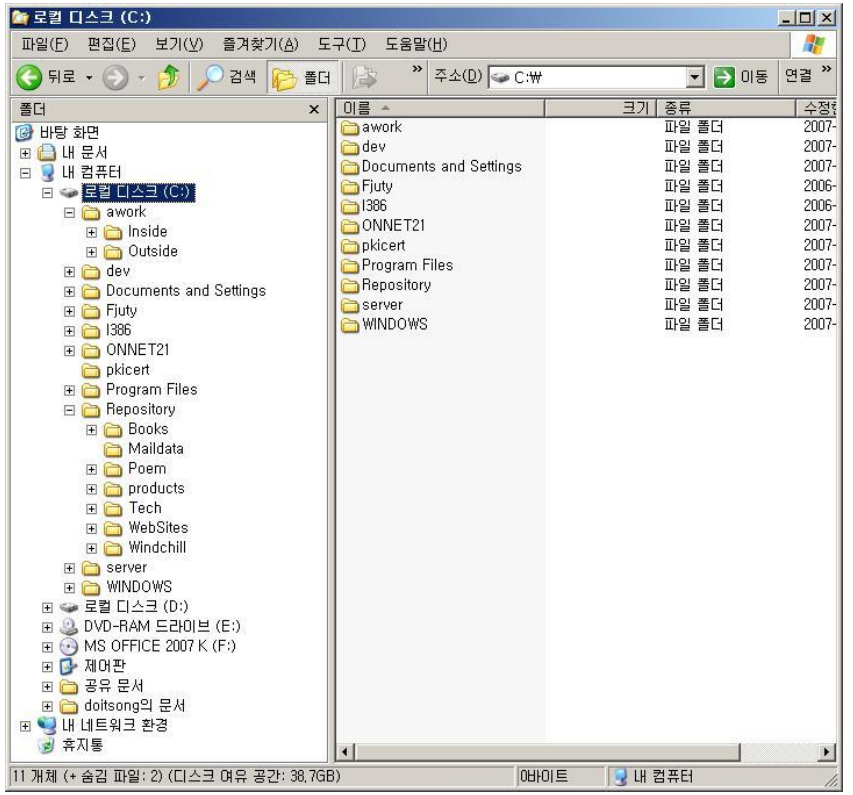
- ✓ 탐색기는 파일시스템의 클라이언트 임
- ✓ 파일 시스템은 커널 내부 또는 커널 위에 존재하는 시스템 서비스

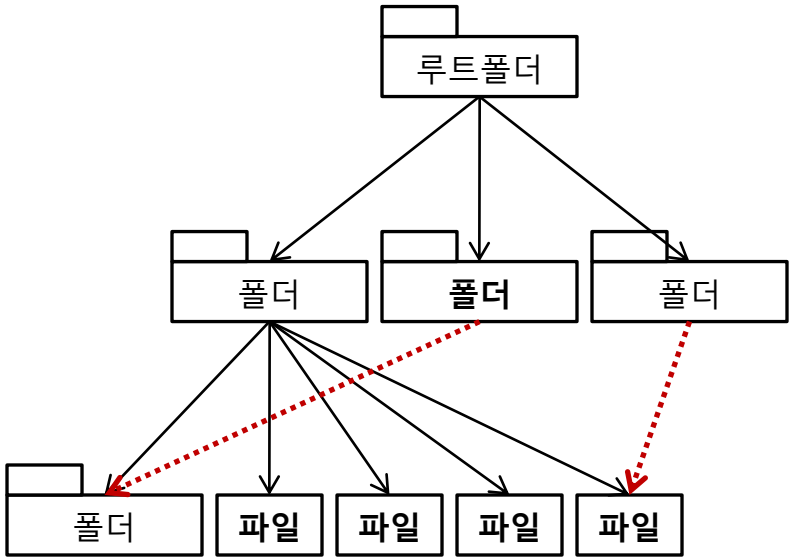




2. 파일 시스템 특징(1/2)

- ✓ 파일시스템의 구조
- ✓ 폴더는 폴더를 가진다.
- ✓ 폴더는 파일을 가진다.
- ✓ 폴더와 파일을 하나의 부모 폴더를 가진다.

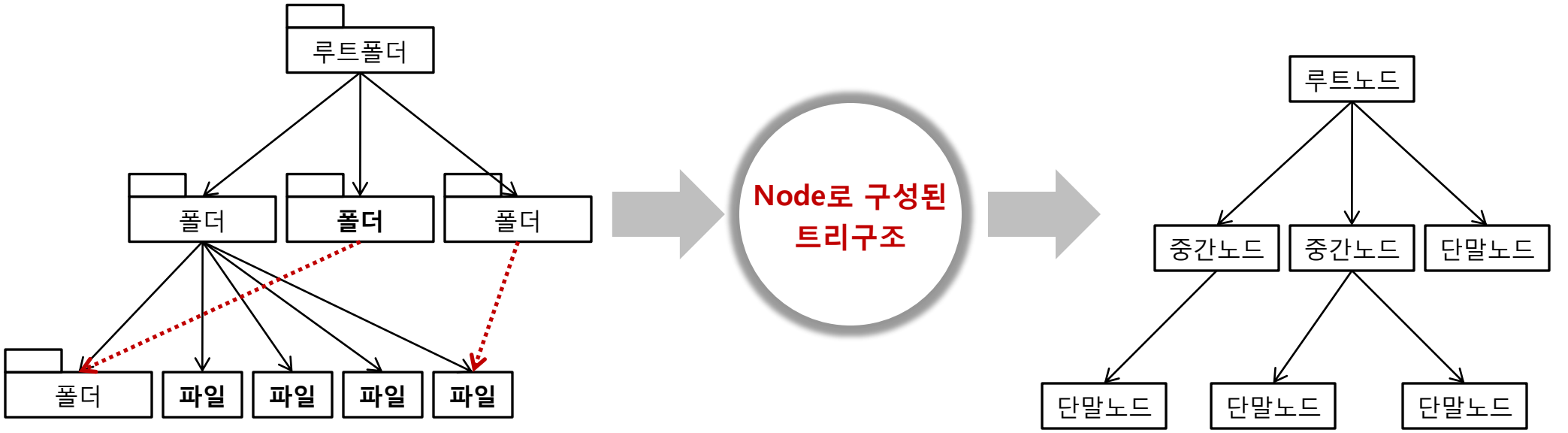




2. 파일 시스템 특징(2/2)

✓ 설계를 위해, 파일 시스템 특징을 추상화하자

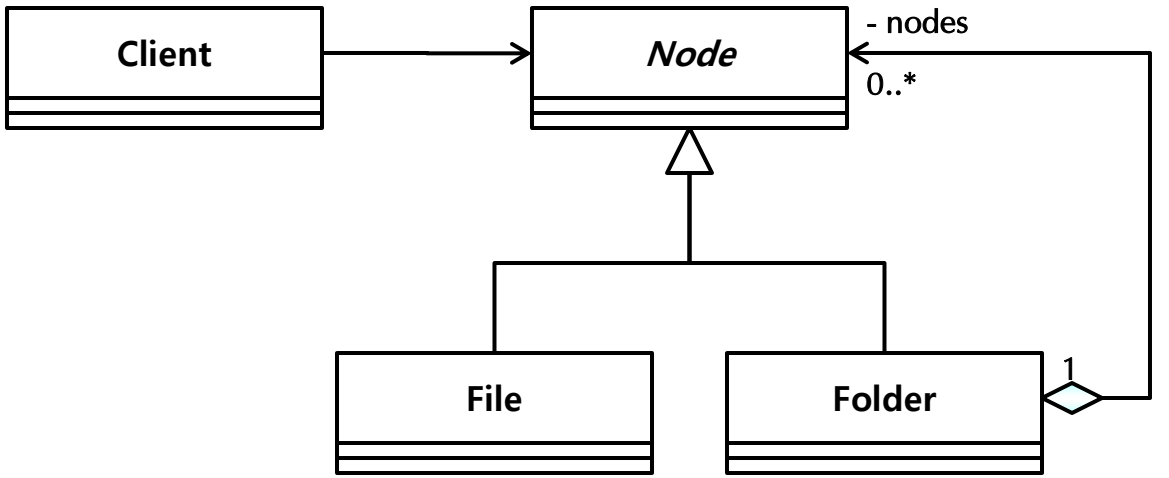
- 루트 존재
- 폴더가 폴더를 가지고
- 폴더가 파일을 가진다.
- 폴더와 파일은 하나의 부모 노드를 가진다.



3. 기본설계 - 트리노드

✓ 추상화 결과

- 파일은 노드이다.
- 폴더도 노드이다.
- 폴더는 노드를 가진다.
- 노드는 하나의 폴더를 가진다.




```
public abstract class Node {
        // 공통 인터페이스
        ....
        protected Node(){
                ...
        }
        ....
}


public class Folder extends Node {
        public Folder() {
                ...
        }
        // 공통 인터페이스 재정의
        private List<Node> nodes;
}


public class File extends Node {
        public File() {
                .....
        }
        // 공통 인터페이스 재정의
}
```

3. 기본 설계 - 노드 인터페이스 정의

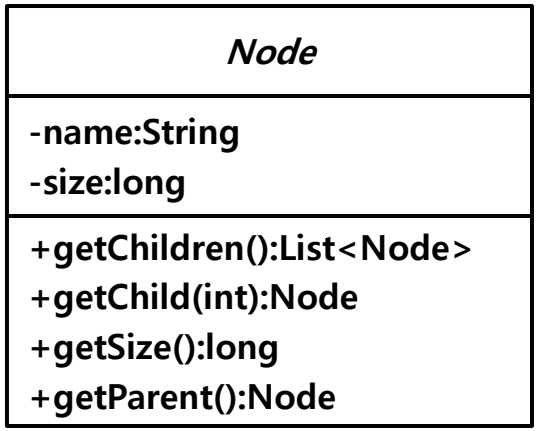
NEXTREE

✓ 속성

- name: 노드의 이름
- size: 노드 크기(바이트 단위)

✓ 오퍼레이션

- getChildren(), getChild(int index): 하위 노드 가져오기
- getSize(): 크기 계산하기
- getParent(): 부모 노드 가져오기



3. 기본 설계 - 트리노드 항해

NEXTREE

✓ 폴더의 자식노드(파일/폴더) 목록을 요청

- 폴더는 자식노드를 열거하기 위한 인터페이스 필요함
- public Node getChild(int index);
- public List<Node> getChildren();
- List<Node>에는 파일과 디렉토리 모두 포함
- 노드(파일/폴더)의 크기를 알려주는 인터페이스 필요함
- public long size();

폴더에서

```
public long size() {
        long total = 0;
        int childCount = nodes.size();
        ...
        for (int i=0; i<childCount; i++) {
                total += getChild(i).size();
        }

        return total;
}
```

파일에서

```
public long size() {
        return this.size;
}
```

4. 적용 패턴 – Composite 패턴

NEXTREE

✓ 의도

- 부분과 전체의 계층을 표현하기 위해 복합 객체를 트리로 구성
- 클라이언트가 개별 객체와 복합 객체를 동일한 방법으로 다룸

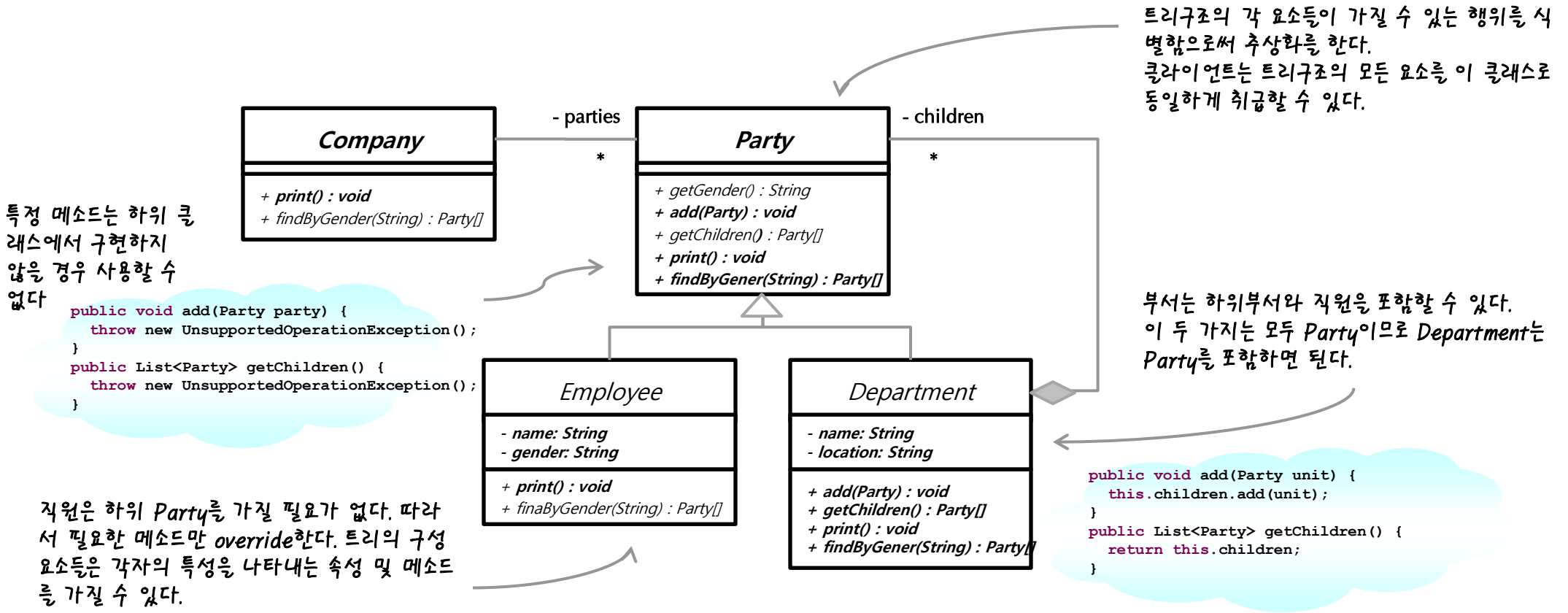
✓ 동기

- 현장에서 만난 문제점을 Composite 패턴을 통해 그 대안을 찾고자 한다.
- 추상화 또는 일반화 하기
- 책임의 위임

4. 적용 패턴 – Composite 패턴

✓ [예제]추상화 또는 일반화 하기

- 부서와 직원은 조직 트리구조의 일부로 동일한 객체로 추상화가 가능
- 클라이언트(회사)가 다루기 쉽도록 동일한 행위를 식별

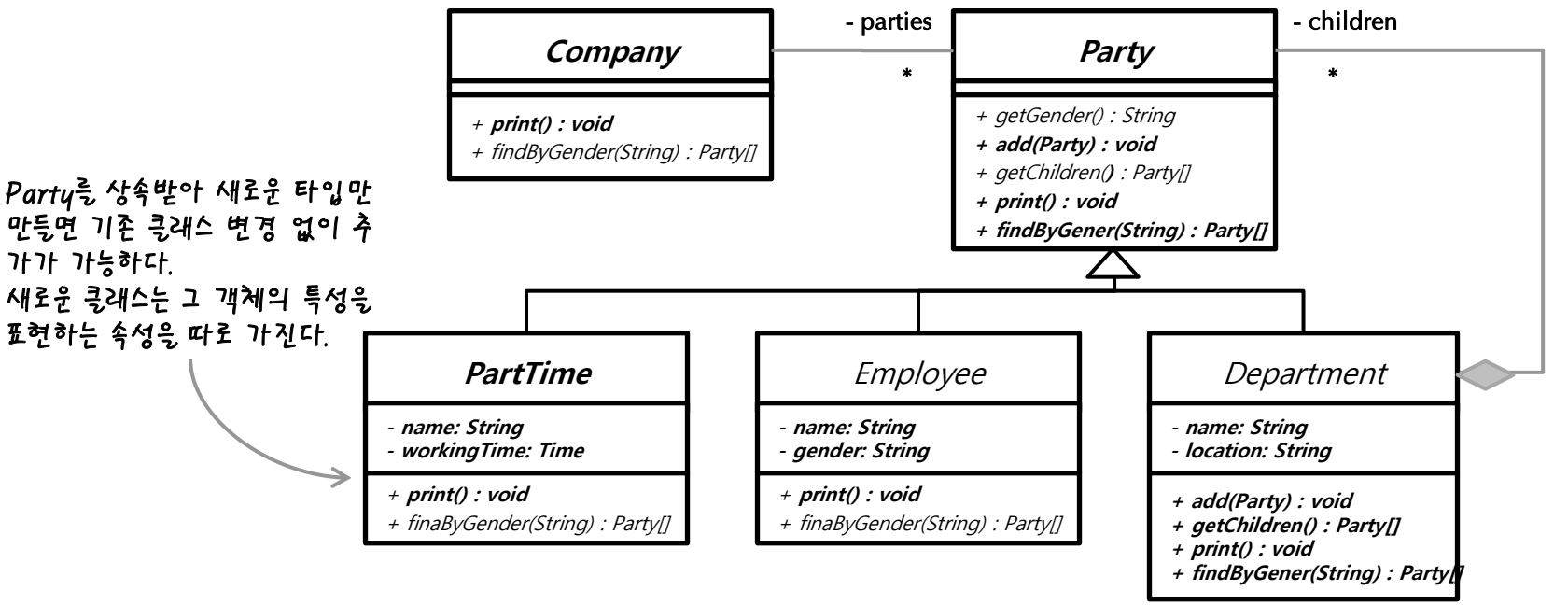


4. 적용 패턴 – Composite 패턴

NEXTREE

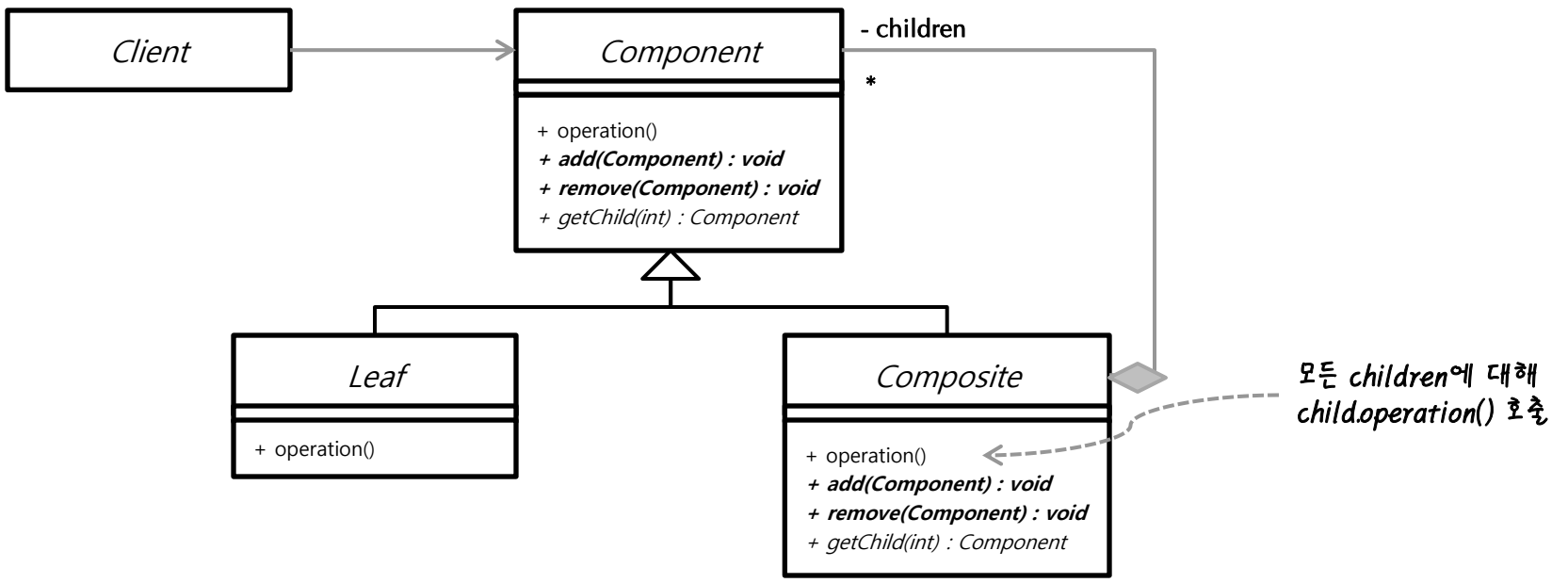
✓ 확장

- 조직 구조에 새로운 타입이 추가될 경우 기존 구조 변경 없이 추가 가능
- Leaf에 해당하는 요소나 Container에 해당하는 요소 모두 추가 용이
  (문제 : Container 확장방법은?)



4. 적용 패턴 – Composite 패턴

✓ 표준 구조



4. 적용 패턴 – Composite 패턴

NEXTREE

✓ 긍정적인 측면
- 임의의 복잡성을 가진 트리 구조를 지원
- 노드의 복잡성이 클라이언트에게 보이지 않음
- 기존의 코드를 그대로 두고 새로운 타입의 컴포넌트를 추가할 수 있음

✓ 부정적인 측면
- 코드가 이해하기 어려워 짐
- 동일한 인터페이스를 강요함

✓ 구현 이슈
- 수행성능을 향상하기 위해 정보를 언제/어디에 캐슁할 것인가?
- 컴포넌트 클래스에 할당할 저장공간은 얼마나?
- Child를 저장하기 위해 어떤 데이터 구조를 사용하는가?
- Child 추가/삭제를 위한 메쏘드가 컴포넌트 클래스에 선언되어야 하는가?

5. 확장 요구사항 - mkdir

NEXTREE

- ✓ 지금까지 작업 결과
  - Composite 패턴을 이용하여 파일시스템의 백본을 설계함
- ✓ 추가 요구사항
  - 클라이언트는 파일과 폴더를 생성하고 필요한 곳에 둘 수 있어야 함
  - 폴더는 child 노드를 가진다(책임이 있음)
  - void attach (Node child);
  - void detach (Node child);

5. 확장 요구사항 - mkdir

- ✓ 현재 폴더= 폴더 객체에 대한 참조
- ✓ mkdir의 절차
  - 새로운 폴더 개체 생성
  - 현재 폴더의 attach()를 호출
  - 새로운 폴더를 패러미터 전달

```
...

Folder currentFolder;
currentFolder.attach (new Folder("newFolder"));
```

5. 확장 요구사항 - mkdir

NEXTREE

- ✓ mkdir subfolderA/subfolderB/newsubfolder의 내부 로직
  - subfolderA 객체를 찾는다.
  - subfolderB 객체를 찾는다.
  - subfolderB 객체가 newsubfolder 객체를 attach() 한다.
- ✓ 클라이언트 클래스 코드

```
public void mkdir (Folder currentFolder, String path) {

        String subpath = path;
        if (subpath == null) {
                currentFolder.attach(new Folder(path));
        } else {
                String name = head(path);
                Node child = find(name, currentFolder);
                if (child != null) {
                        mkdir(child, subpath);
                } else {
                        // error process
```

5. 확장 요구사항 - mkdir

- ✓ Child 개체의 성격
  - 디렉토리 객체인가 파일 객체인가?
  - 파일 객체일 경우 방문할 필요 없음
- ✓ 따라서,

```
if (node != null) {
        try {
                Folder child = (Folder)node;
                //java.lang.ClassCastException(파일 객체일 경우)
                mkdir(child, subpath)
        } catch (java.lang.ClassCaseException cce) {
                System.out.println(" 폴더가 아닙니다.");
                .....
```

6. 확장 요구사항 - symbolic link

NEXTREE

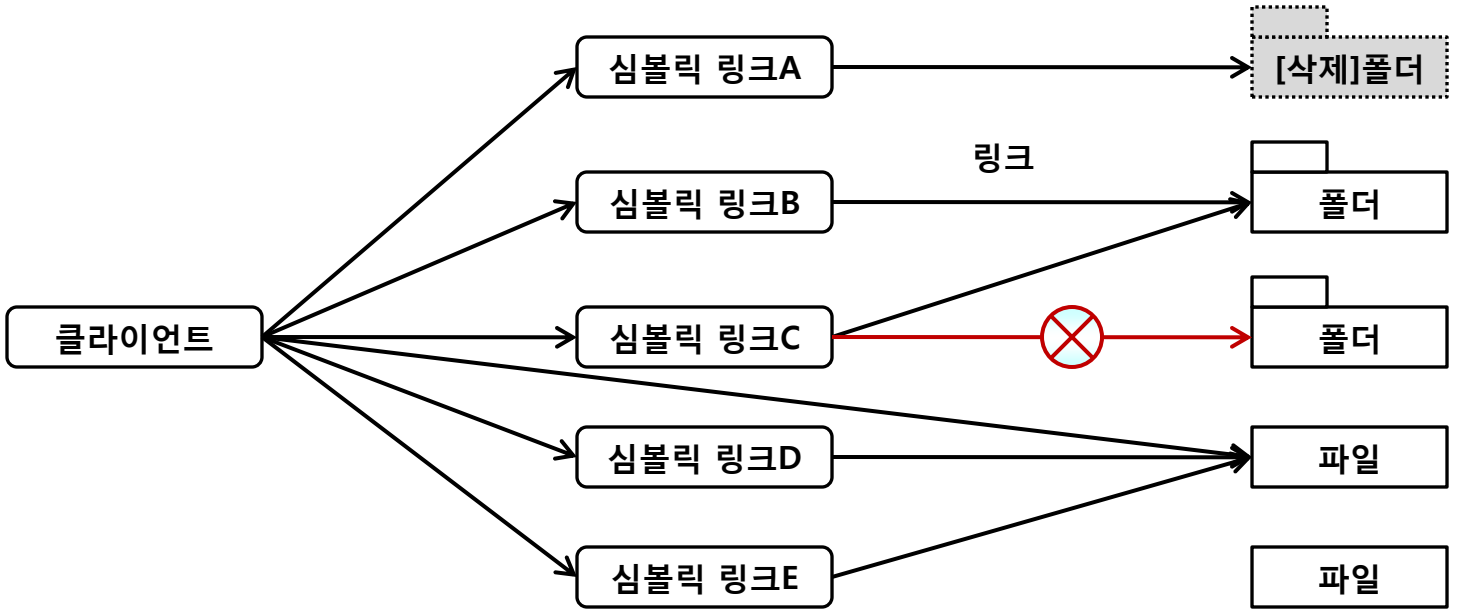
- ✓ 사용 예
  - Symbolic link : Unix
  - Aliases : Mac Finder parlance
  - Shortcuts : Windows 95
- ✓ 파일이나 디렉토리에 대한 링크
  - 파일 링크 : 편집, 저장
  - 디렉토리 링크 : 노드 추가 삭제

6. 확장 요구사항 - symbolic link

NEXTREE

✓ Symbolic link의 개념

- 대상은 폴더, 파일 모두 가능
- 하나의 심볼릭 링크는 하나의 대상만을 가짐
- 하나의 대상에 대해 여러 개의 심볼릭 링크를 만들 수 있음
- 대상이 지워져도 심볼릭 링크는 존재함



6. 확장 요구사항 - symbolic link

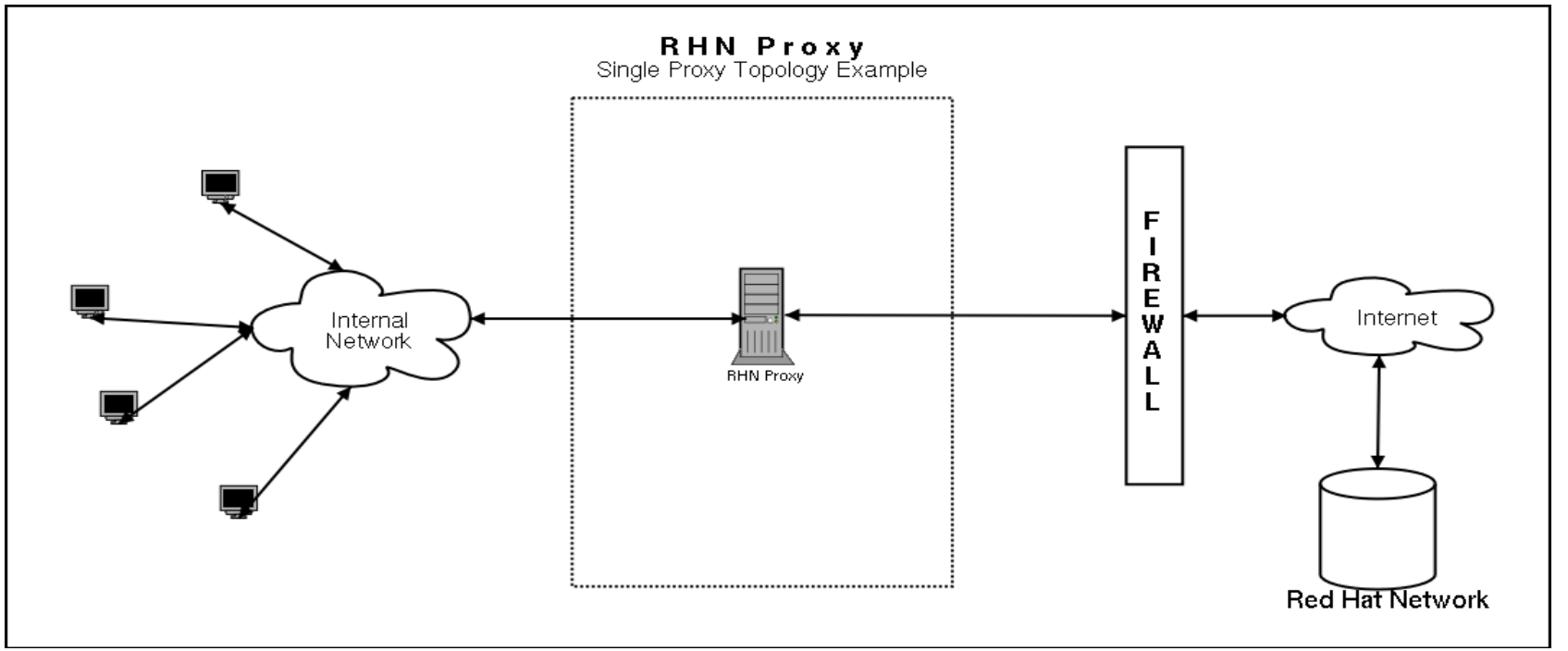
목적	디자인 패턴	특징
생성 패턴	Abstract Factory	상품 객체들의 패밀리
	Builder	복합 객체 생성
	Factory Method	인스턴스화될 객체의 하위 클래스
	Prototype	인스턴스화 될 객체의 클래스
	Singleton	단일 인스턴스
구조 패턴	Adapter	객체를 위한 인터페이스
	Bridge	객체의 구현
	Composite	객체구조와 객체의 복합체
	Decorator	상속없이 객체에 책임추가
	Facade	하위 시스템의 인터페이스
	Flyweight	객체 저장 비용
	Proxy	객체의 위치와 접근방법
행위 패턴	Chain of Responsibility	요청을 처리할 수 있는 객체들
	Command	요청처리 시점과 방법
	Interpreter	언어의 문법과 해석
	Iterator	집합요소에 접근하고 횡단함
	Mediator	어떤 객체가 어떻게 다른 객체와 상호작용하는가?
	Memento	어떤 사적정보를 언제 객체의 밖에 저장하는가?
	Observer	의존 객체 업데이트 되는 방법
	State	객체의 상태
	Strategy	알고리즘 선택
	Template Method	알고리즘 처리 절차
	Visitor	오퍼레이션 추가

적절한 패턴 찾기

7. 적용 패턴 – Proxy 패턴

✓ 주변에서 찾을 수 있는 예, Proxy 서버

- 컨텐츠 캐싱
- 컨텐츠 접근 통제
- 원격 서버의 대리인 역할
- Proxy == Agent



7. 적용 패턴 – Proxy 패턴

NEXTREE

✓ 의도

- 다른 객체에 접근하기 위해 중간 대리 역할을 하는 객체를 사용

✓ 동기

- 심볼릭 링크 문제점을 Proxy 패턴을 통해 그 대안을 찾고자 한다.
- 폴더와 파일에 대한 접근 통제
- 지연 로딩 (Lazy Loading)
- 영향도 최소화

7. 적용 패턴 – Proxy 패턴

NEXTREE

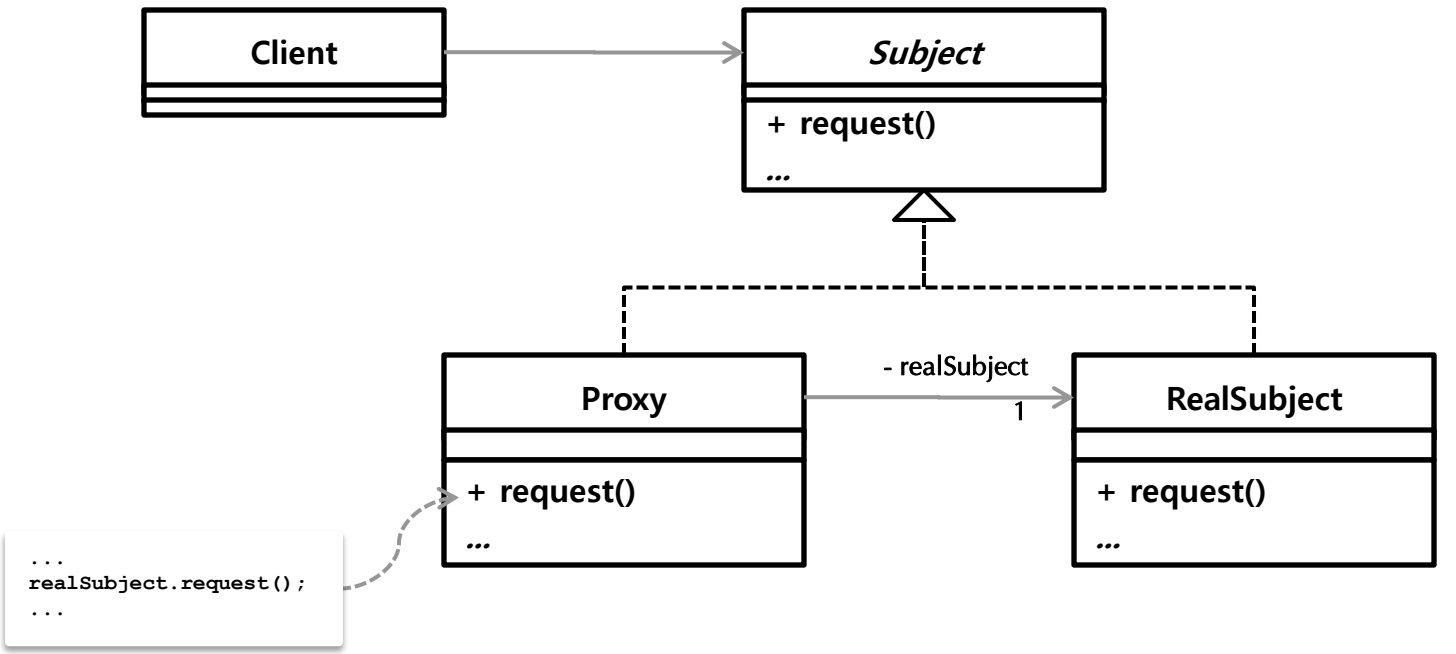
✓ 의도
- 다른 객체에 접근하기 위해 중간 대리 역할을 하는 객체를 사용

✓ 동기
- 심볼릭 링크 문제점을 Proxy 패턴을 통해 그 대안을 찾고자 한다.
- 폴더와 파일에 대한 접근 통제
- 지연 로딩 (Lazy Loading)
- 영향도 최소화

✓ 적용
- 원격 호출이 필요할 경우 클라이언트는 원격 Proxy를 통해 호출 (예: RMI)
- 요청이 있을 경우에만 복잡한 객체를 생성할 필요가 있을 경우
  (성능이 떨어지거나 메모리를 많이 사용하거나)
- 원래 객체에 대한 접근을 제어할 경우 보호용 Proxy를 사용

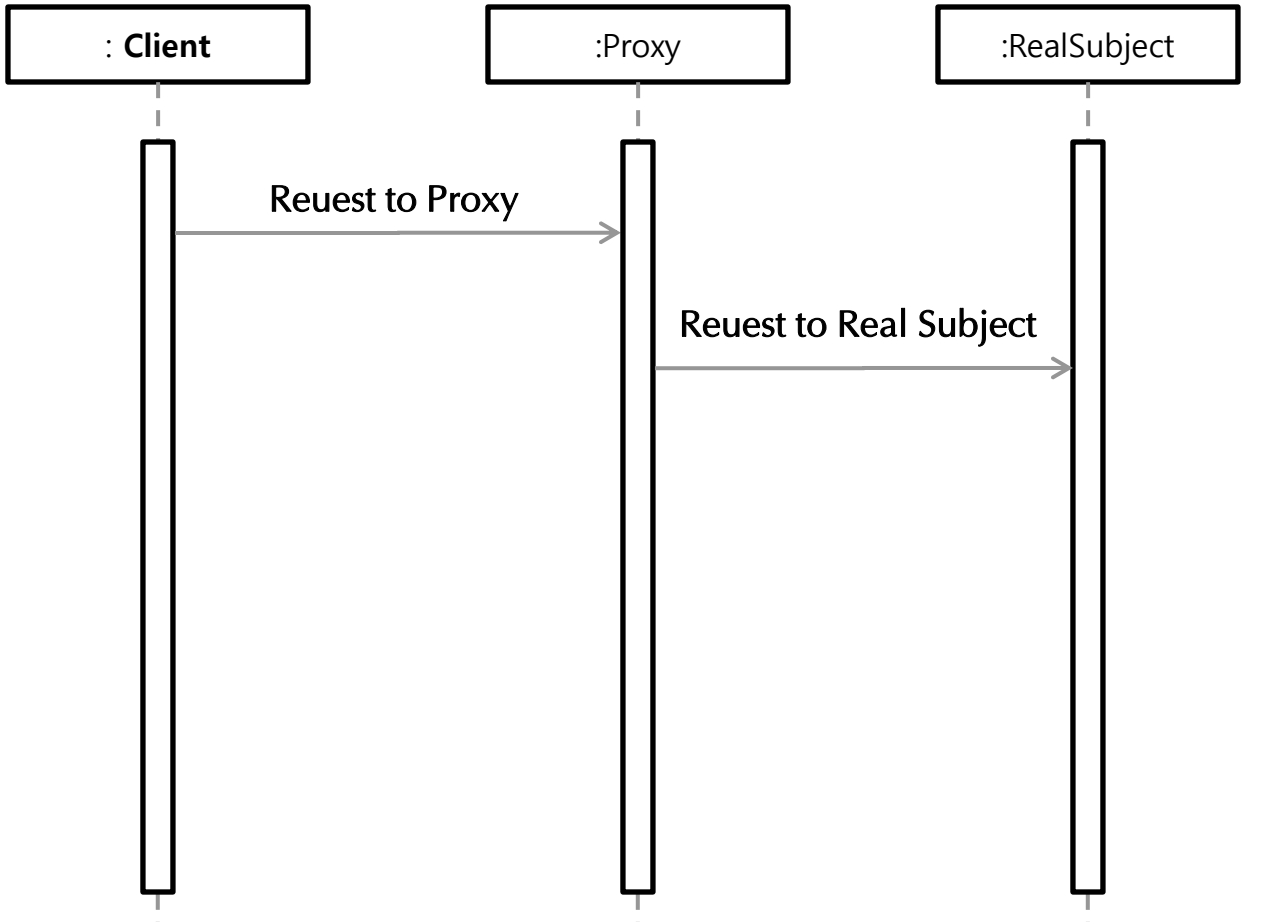
7. 적용 패턴 – Proxy 패턴

✓ 표준 구조



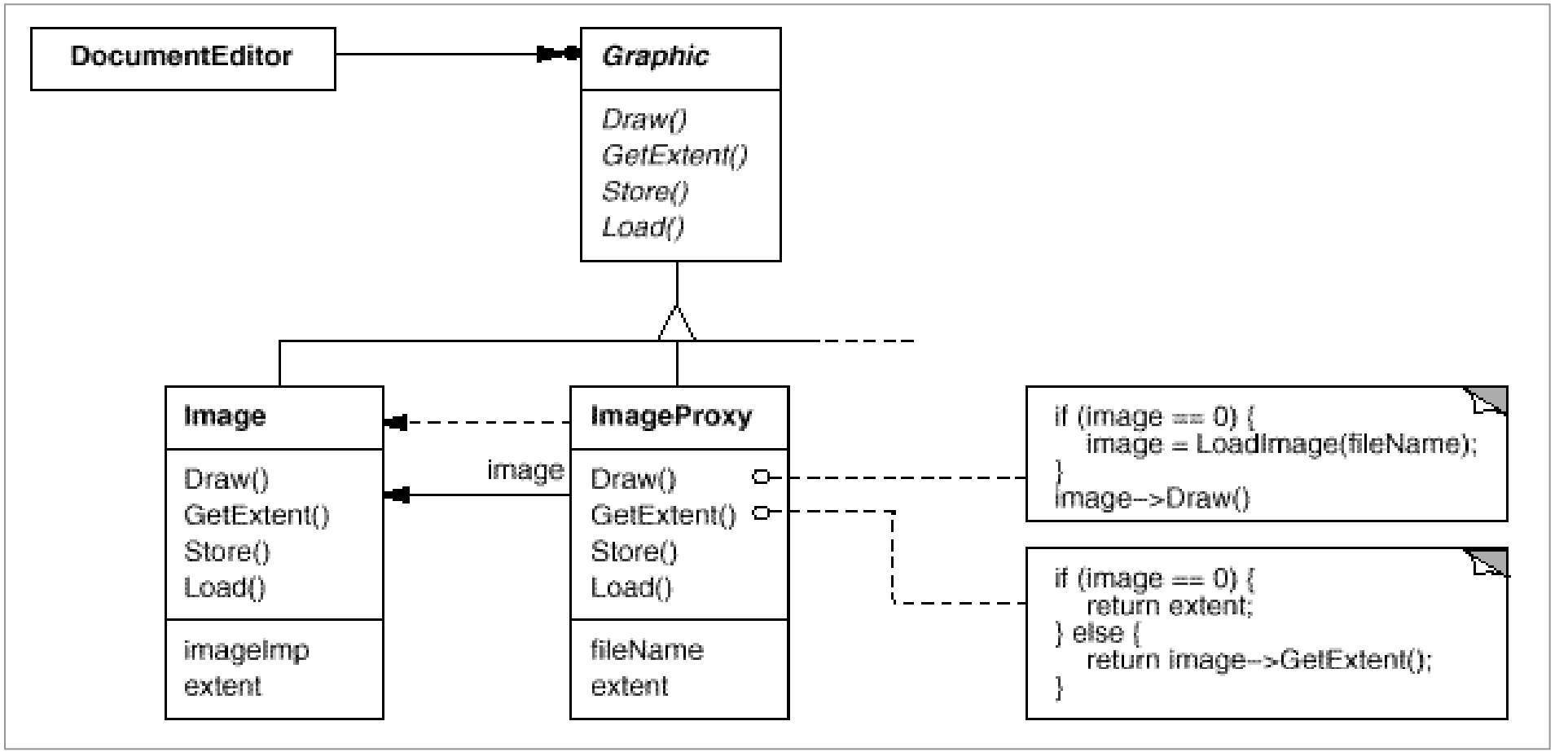
7. 적용 패턴 – Proxy 패턴

✓ 표준 흐름



7. 적용 패턴 – Proxy 패턴

✓ Proxy 패턴 예제



8. 심볼릭 링크 설계

✓ 링크 소스 코드

```
public class Link extends Node {
	public Link(Node node) {
		//
		....
	}
	// 실제 객체(파일 또는 디렉토리)에 대한 참조
	private Node realSubject;
}
```

8. 심볼릭 링크 설계

NEXTREE

- ✓ Proxy의 실제 주체(real subject)
  - 여기서는 파일과 폴더가 실제 주체(realSubject)
  - [GOF]에서는
  - Proxy는 실제 객체에 접근하기 위한 참조(포인터)를 유지한다.
  - realSubject는 대상 Node에 대한 포인터
  - 공통 인터페이스 없이 파일과 폴더 모두에 적용되는 symbolic link를 만들기 어려움

8. 심볼릭 링크 설계

NEXTREE

✓ 소스코드에서 real subject의 용도

- Link 클래스의 소스코드에서

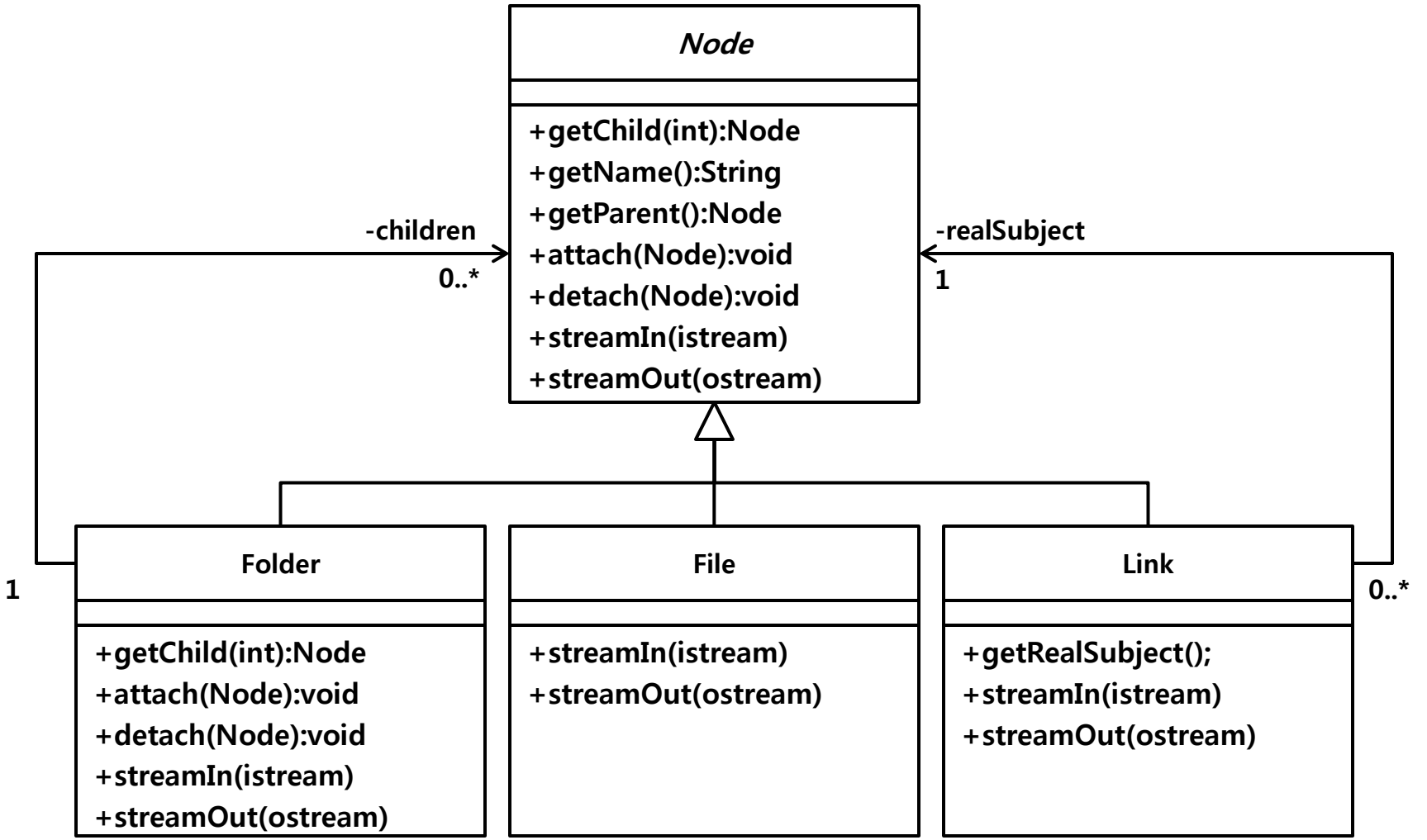
```
....
        public Node getChild(int index) {
                // realSubject 로 위임
                return realSubject.getChild(index);
        }
```

8. 심볼릭 링크 설계

- ✓ Proxy 패턴 적용 시 고려사항
  - 파일이 삭제되었을 경우
  - Proxy는 허상을 참조하게 된다(dangling pointer)
  - Proxy에게 삭제 사실을 알려줄 Observer가 필요하다.
- ✓ 유닉스와 매킨토시, Symbolic link는 참조하고 있는 대상 파일의 실제 이름을 가지고 있다.
- ✓ 따라서 Proxy는
  - 파일 시스템의 root에 대한 참조와 파일의 이름을 가지고 있다.
  - 이름을 찾는(lookup) 비용이 높음
- ✓ 두가지 해결책
  - 이름 찾는 비용을 감수하고 이름과 참조를 가지고 있음
  - Observer에 대한 포인터를 가지고 있음

8. 심볼릭 링크 설계

✓ 심볼릭 링크를 반영한 클래스 다이어그램



9. 확장 요구사항 – cat, ls

NEXTREE

✓ 요구사항이 계속 식별됨

- 추가 속성 지원, 파일 생성일자, 최근 변경일자 등
- 새로운 유형의 size 계산 지원, 블록(512바이트) 단위 크기 계산
- 노드를 위한 아이콘 리턴
- 결과적으로, Node는 거대한(giant)클래스가 되어 감
- 인터페이스 추가 없이 기능을 추가하여야 하는 상황임
- 이러한 문제를 어떻게 해결하는가?

9. 확장 요구사항 – cat, ls

NEXTREE

- ✓ 단순한 기능 추가
  - 파일에 포함된 단어 개수 리턴
  - getWordCount()
  - 노드 클래스에 추가?
  - 파일 클래스에 추가?
  - 그러면
  - Stream 인터페이스가 있으므로
  - stream 인터페이스를 활용하여 클라이언트 단에서 getWordCount()를 구현한다.
- ✓ 충분한 Primitive 인터페이스 준비
- ✓ Primitive 인터페이스의 조합을 통해 원하는 기능을 클라이언트 측에서 구현

9. 확장 요구사항 – cat, ls

NEXTREE

- ✓ 주요 기능, cat 추가
  - 유닉스의 cat 명령어
  - 파일과 디렉토리는 명령의 결과가 다름
  - 따라서 개별적으로 구현이 되어야 함
  - 기존의 코드를 바꾸는 것이 바람직하지 않음
- ✓ 이 경우, 클라이언트 코드는

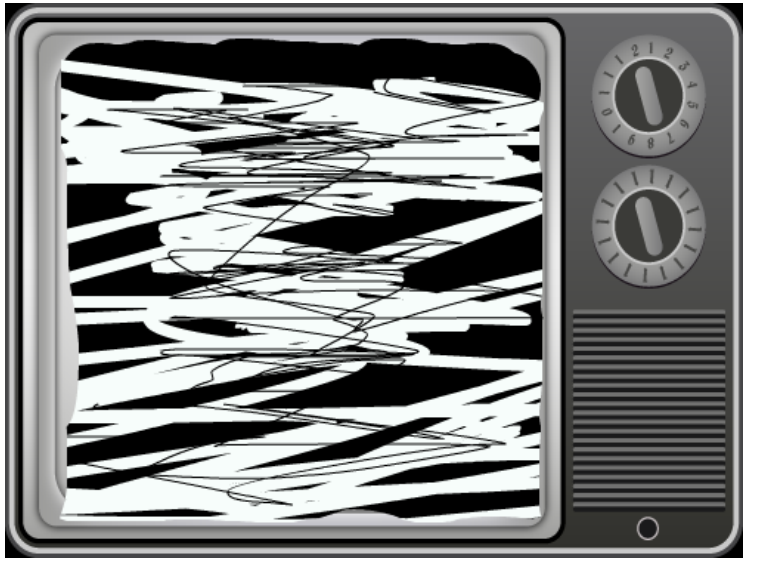
새로운 노드가
생길 경우 ???

```
public void cat(Node node) {
        Link l;
        if class is file              // 다운캐스팅
                do something;
        else if class is Directory    // 다운캐스팅
                do something;
        else if class is Link         // 다운캐스팅
                do somthing;
        .....
```

9. 확장 요구사항 – cat, ls

✓ 실세계의 예를 들어봅시다.

- 나른한 주말 오후,
- 아이랑 놀다가 한 잠 자고 싶은데,
- TV를 고쳐달라고 하는 아내의 성화가 계속되고...



[출처: http://sketchpan.com/?guest=54857]

9. 확장 요구사항 – cat, ls

목적	디자인 패턴	특징
생성 패턴	Abstract Factory	상품 객체들의 패밀리
	Builder	복합 객체 생성
	Factory Method	인스턴스화될 객체의 하위 클래스
	Prototype	인스턴스화 될 객체의 클래스
	Singleton	단일 인스턴스
구조 패턴	Adapter	객체를 위한 인터페이스
	Bridge	객체의 구현
	Composite	객체구조와 객체의 복합체
	Decorator	상속없이 객체에 책임추가
	Facade	하위 시스템의 인터페이스
	Flyweight	객체 저장 비용
	Proxy	객체의 위치와 접근방법
행위 패턴	Chain of Responsibility	요청을 처리할 수 있는 객체들
	Command	요청처리 시점과 방법
	Interpreter	언어의 문법과 해석
	Iterator	집합요소에 접근하고 횡단함
	Mediator	어떤 객체가 어떻게 다른 객체와 상호작용하는가?
	Memento	어떤 사적정보를 언제 객체의 밖에 저장하는가?
	Observer	의존 객체 업데이트 되는 방법
	State	객체의 상태
	Strategy	알고리즘 선택
	Template Method	알고리즘 처리 절차
	Visitor	오퍼레이션 추가

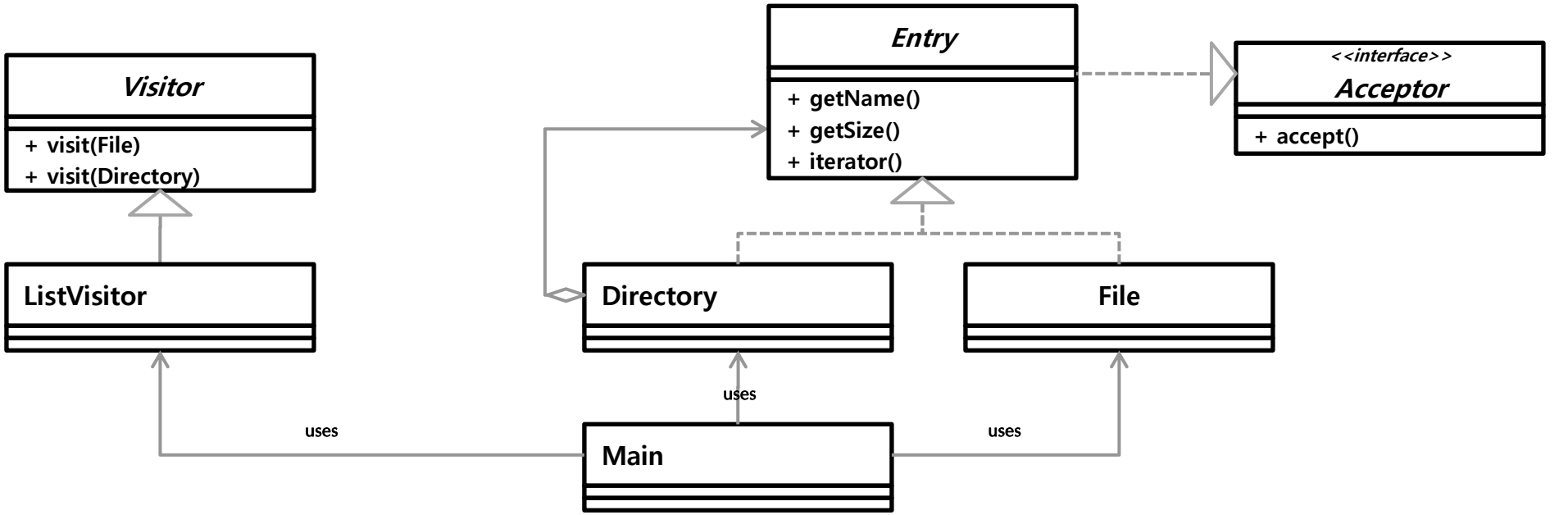
10. 적용 패턴 – Visitor 패턴

✓ 의도

- 객체 구조에 속한 요소에 수행될 오퍼레이션을 정의하는 객체이다. Visitor 패턴은 처리되어야 하는 요소에 대한 클래스를 변경하지 않고 새로운 오퍼레이션을 정의할 수 있게 한다.

✓ 동기

- 파일과 디렉토리로 구성된 데이터 구조 안을 방문자가 찾아 다니는 파일의 일람을 표시하는 예제입니다.



10. 적용 패턴 – Visitor 패턴

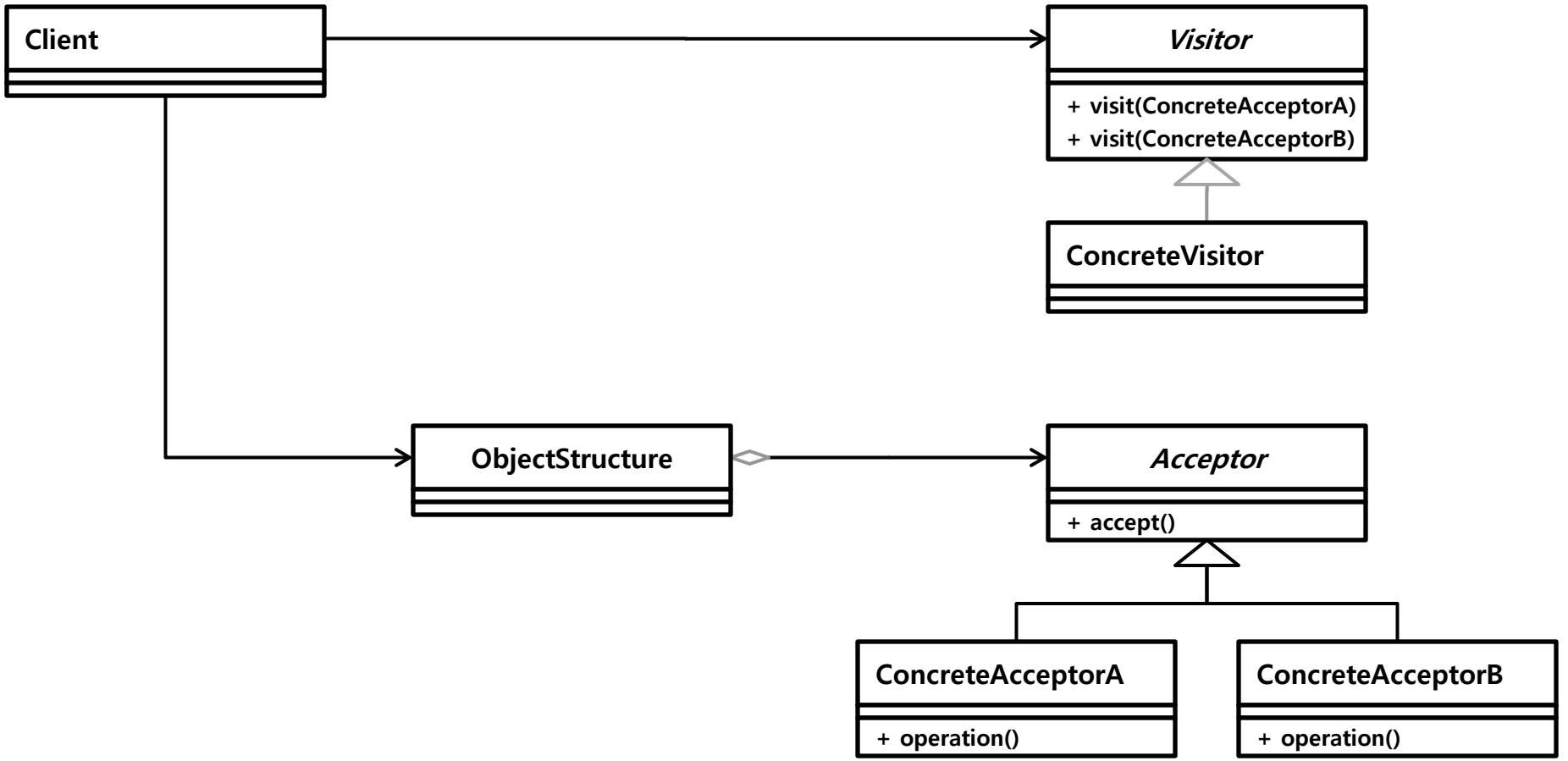
NEXTREE

✓ 적용

- 객체 구조가 다른 인터페이스를 가진 클래스들을 포함하고 있어서 구체적 클래스에 따라 이들 오퍼레이션을 수행하고자 할 때.
- 구별되고 관련되지 않은 많은 오퍼레이션이 객체에 수행될 필요가 있지만 오퍼레이션으로 인해 클래스들을 복잡하게 하고 싶지 않을 때. Visitor클래스는 하나의 클래스에 관련되지 않은 모든 오퍼레이션을 함께 정의함으로써 관련된 오퍼레이션을 유지시킨다.
- 객체 구조를 정의한 클래스는 거의 변하지 않지만 전체 구조에 걸쳐 새로운 오퍼레이션을 추가하고 싶을 때 객체 구조를 변경하려면 모든 Visitor 인터페이스를 재정의해야 한다.

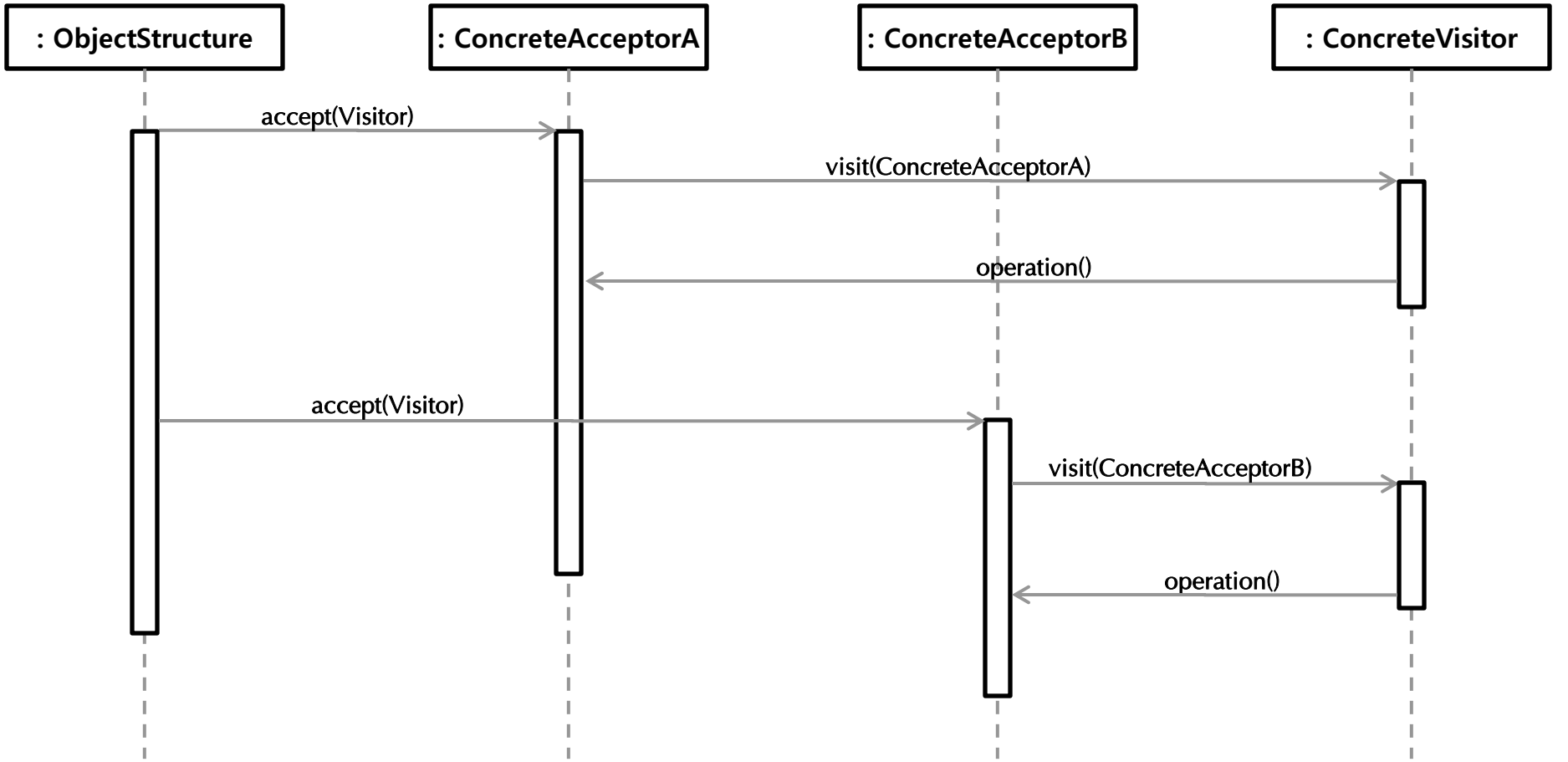
10. 적용 패턴 – Visitor 패턴

✓ 표준 구조



10. 적용 패턴 – Visitor 패턴

✓ 호출 구조



11. ls, cat 설계

- ✓ Visitor 패턴 적용
- ✓ Node 클래스에서
  - abstract void accept (Visitor);
- ✓ 각 구체적인 하위클래스에서
  - 파일 : public void accept(Visitor v) {v.visit(this);}
  - 폴더 : public void accept(Visitor v) {v.visit(this);}
  - 링크 : public void accept(Visitor v) {v.visit(this);}
- ✓ Visitor 클래스

```
public class Visitor {
        public Visitor() {…}
        public void visit(File file) {file.streamOut(cout);}
        public void visit(Folder dir) { some message….; }
        public void visit(Link link) {link.getSubject().accept(this);}
}
```

11. ls, cat 설계

- ✓ 하위 노드 방문(visit)
  - 클라이언트 코드
  - Visitor catVisitor = new CatVisitor();
  - node.accept(catVisitor);
- ✓ 또 다른 기능 추가 : ls

11. ls, cat 설계

NEXTREE

✓ LsVisitor

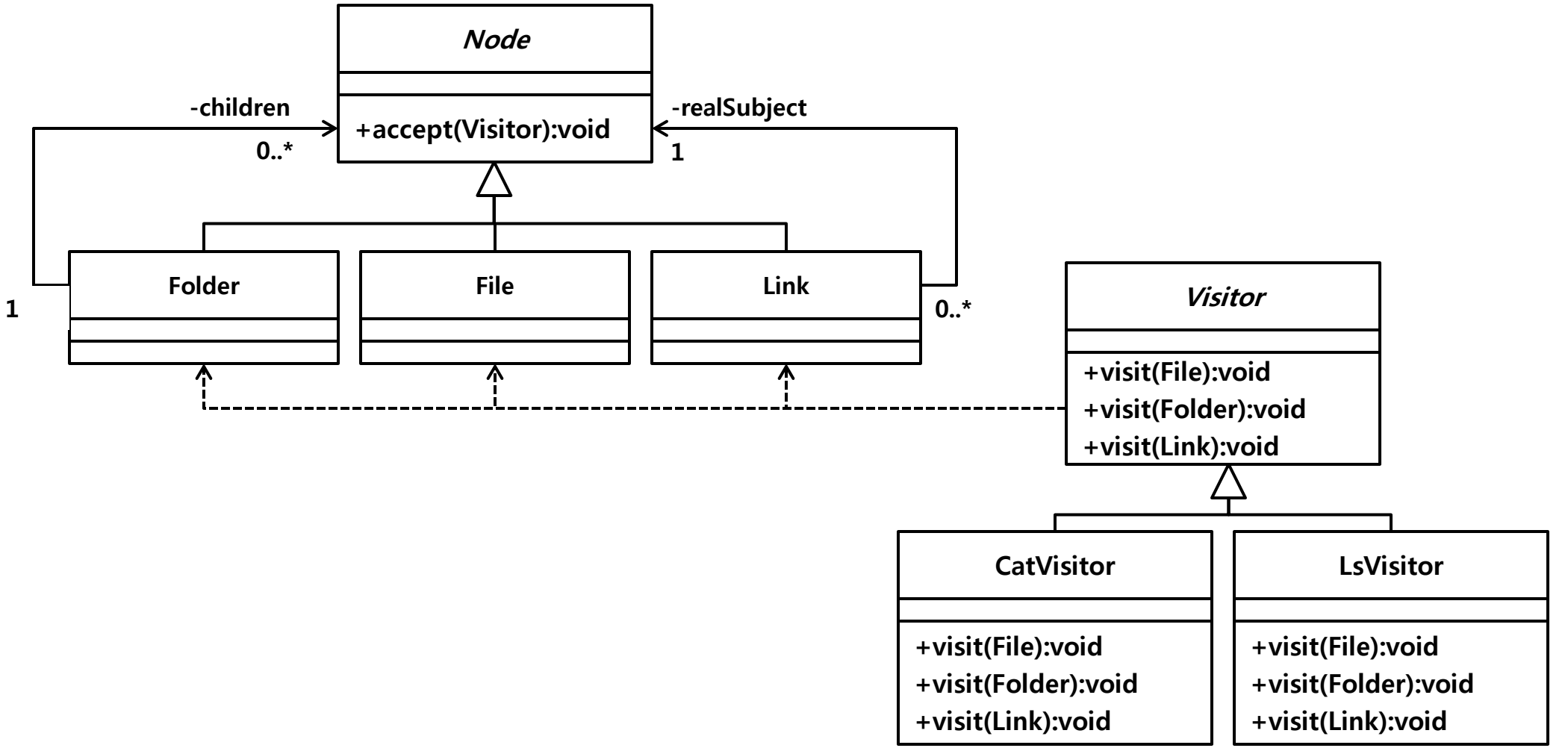
```
class LsVisitor extends Visitor {
        public LsVisitor() { ...}
        public void visit(File file) { ;}
        public void visit(Folder dir) { System.out.print( "/"); }
        public void visit(Link link) {System.out.print("@"); }
};
```

✓ 클라이언트

```
public void ls(Node node) {
    LsVisitor lsv = new LsVisitor();
    Node child;
    for (int I=0; node.size(); I++) {
        child = node.getChild(I);
        System.out.print(child.getName());
        child.accept(lsv);
        System.out.println(" ");
    }
```

11. ls, cat 설계

✓ Visitor 패턴을 활용한 ls, cat 설계 모델



12. 질의 응답 및 토론

- ✓ 질의 응답
- ✓ 토론

❖넥스트리소프트(주)
❖송태국 (tsong@nextree.co.kr)
❖부사장/CTO