Decorator

Decorator - 문제

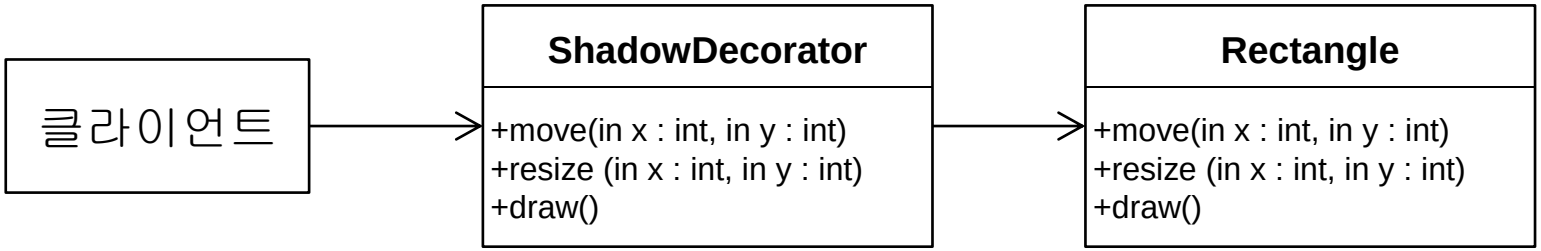
- 어떤 클래스에 기능을 추가하려고 함.

- 그 클래스 소스 코드를 수정하지 않고 기능을 추가해야 한다.
  - 그 클래스 소스 코드를 수정할 수 없어서.
  - 추가하려는 기능이, 그 클래스와 유지 보수 이유가 달라서.

- 어떤 클래스 소스 코드를 수정하지 않고, 기능을 추가하는 방법은?

Decorator – 위임 구조

- Rectangle 클래스를 수정하지 않고, 그림자 그리는 기능 추가
- 그림자 그리는 기능을 wrapper 클래스에 구현



Decorator – 위임 구조

- 그래픽 에디터의 명령을
  - 장식 객체가 받아서 먼저 처리하고
  - 도형 객체에 전달한다

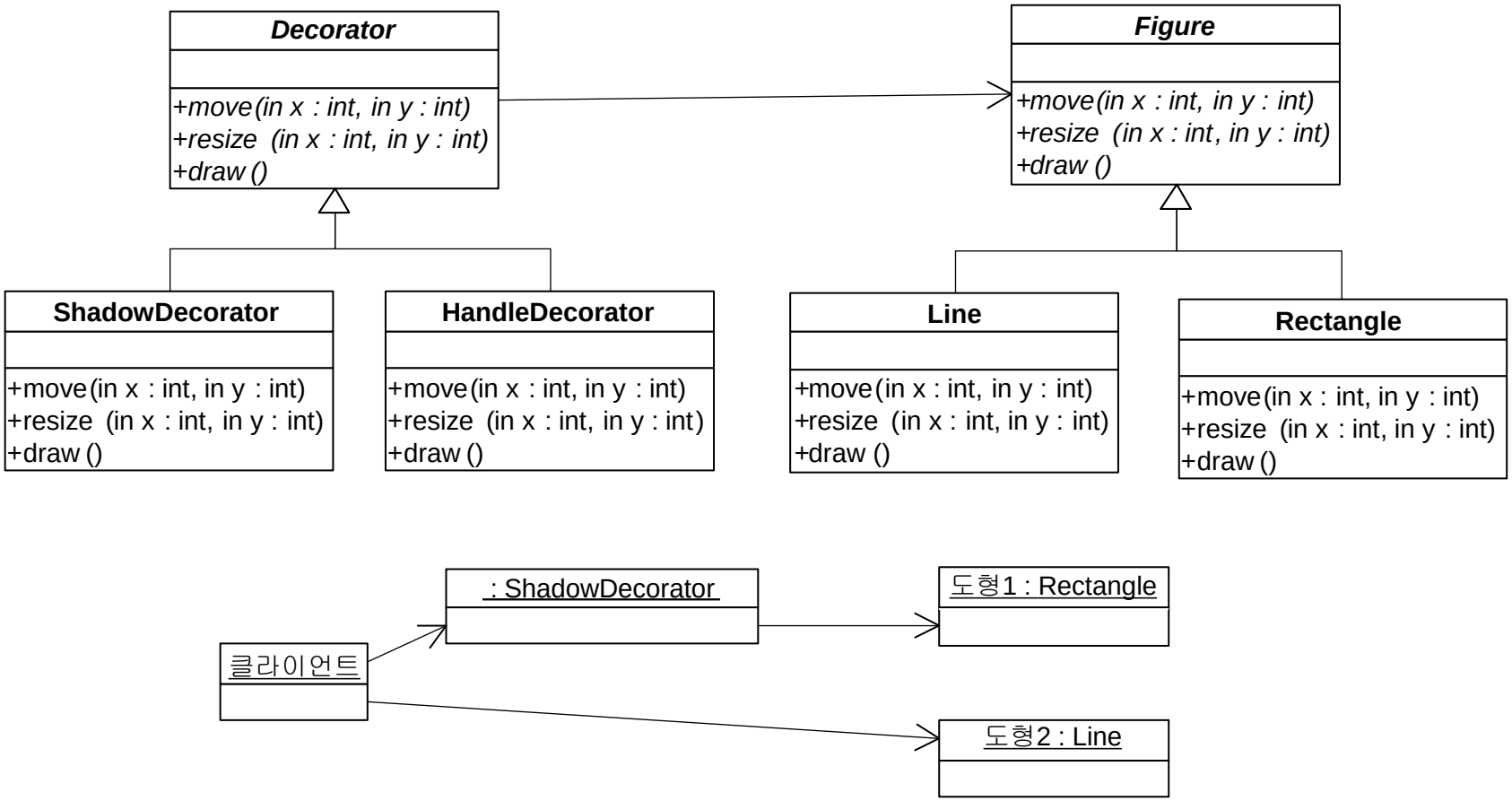
- 즉 그래픽 에디터가 그리라는 명령을 보내면
  - 장식 객체의 draw() 메소드가 먼저 호출되어
  - 장식 객체를 그리고
  - 도형 객체의 draw() 메소드가 호출되어
  - 도형 객체를 그린다
  - 결국 장식 객체와 도형 객체 둘 다 그려진다

Decorator – 위임 구조

- 장식 객체는 draw() 메소드 뿐만 아니라 Figure의 모든 메소드를 구현해야 한다

- 예를 들어 그래픽 에디터가 이동 명령을 보내면
  - 장식 객체의 move() 메소드가 먼저 호출되어
  - 장식 객체가 이동하고
  - 도형 객체의 move() 메소드가 호출되어
  - 도형 객체가 이동한다
  - 결국 장식 객체와 도형 객체 둘 다 이동한다

Decorator – 위임 구조

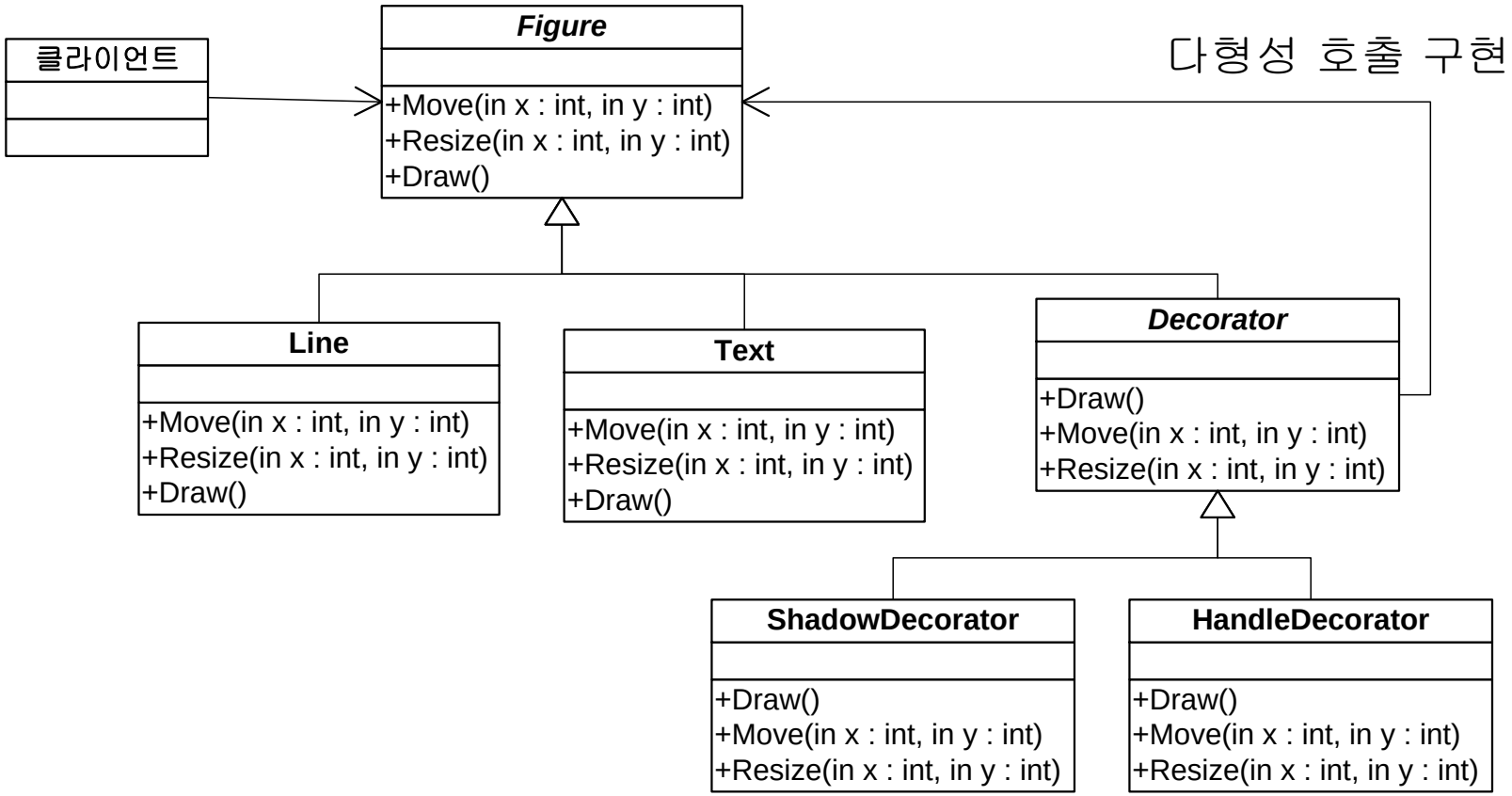


남은 문제

- 클라이언트가 figure, decorator 둘 다 참조해야 하나?

- 클라이언트 소스코드에 decorator에 대한 참조 없이,
  앞 슬라이드의 객체 구조가 가능하려면?

Decorator – 구조



Decorator – 결과

- decorator 를 여러개 결합하여 사용할 수 있다
  - 예: ShadowDecorator → HandleDecorator → Line
- decorator 기능과 무관하게 도형이 추가될 수 있다
- 도형과 무관하게 decorator 가 추가될 수 있다
- 클라이언크는 decorator와 무관하게 구현될 수 있다
- decorator 패턴과 proxy 패턴의 공통점과 차이점은??

예제 코드 분석

- composite.f2 예제에 decorator 패턴 구현
  → decorator.e2

예제 코드 분석

- decorator.i1 예제

- 조건식을 만족하는 항목만 탐색하는 iterator 구현
  - MyArrayConditionalIterator
  - MyArrayConditionalReverseIterator
  - MyListConditionalIterator
  - MyListConditionalReverseIterator

- 유사한 코드 중복이 많다.
  - 이 코드 중복을 어떻게 제거?

구현 실습

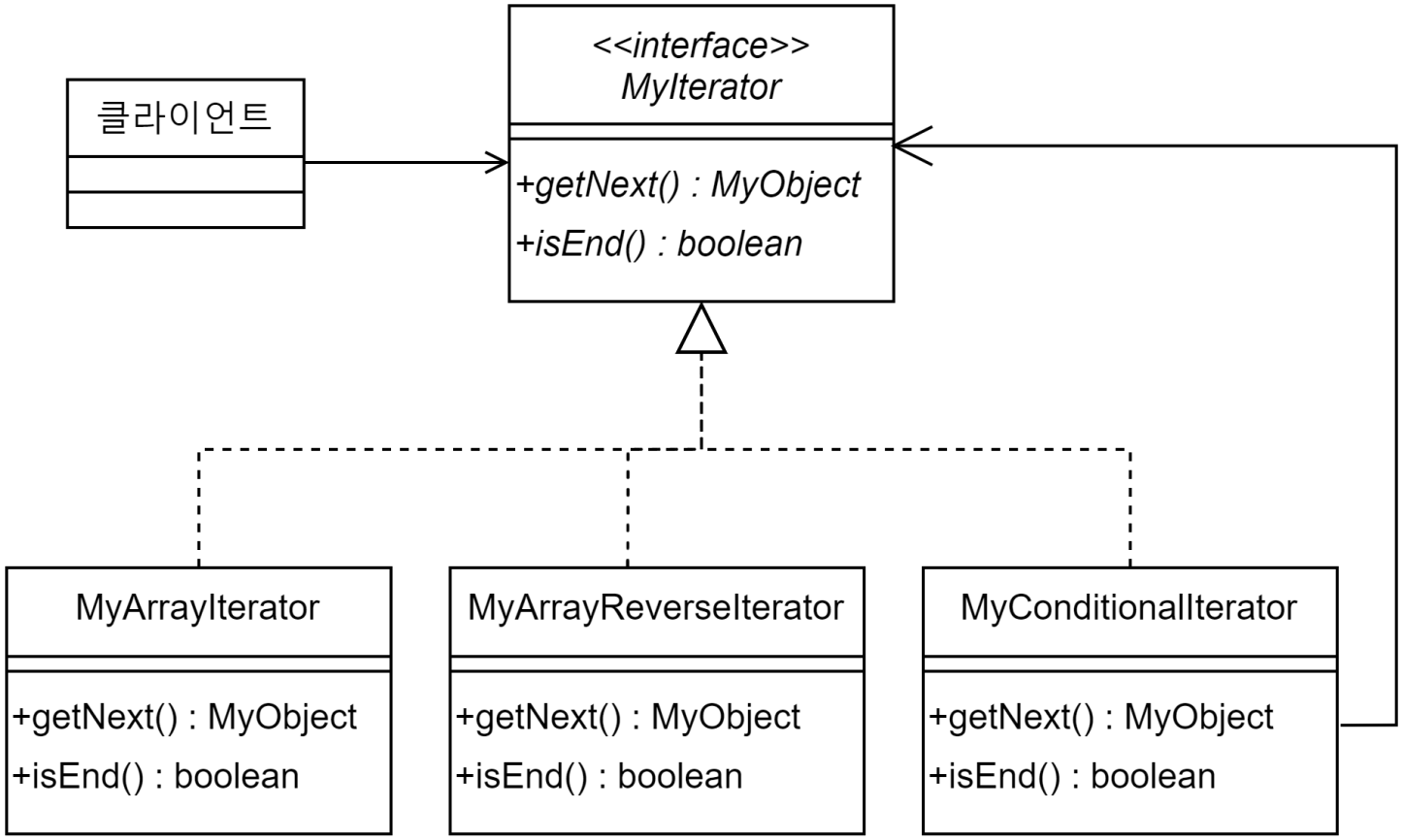
- decorator i1 예제에 decorator 패턴 적용

- MyConditionalIterator
  - 주어진 조건식을 만족하는 항목만 탐색
  - decorator 패턴의 iterator

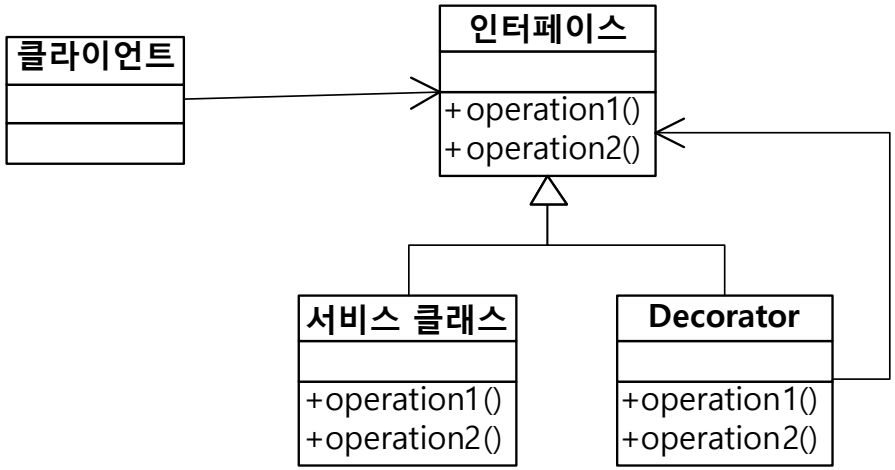
- MyConditionalIterator + MyArrayIterator
  - MyArray 목록들 중 주어진 조건식을 만족하는 항목만 순방향 탐색

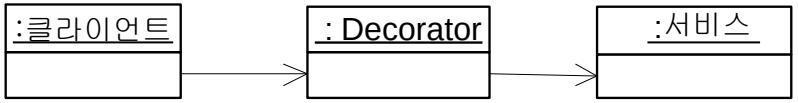
- MyConditionalIterator + MyArrayReverseIterator
  - MyArray 목록들 중 주어진 조건식을 만족하는 항목만 역방향 탐색

구현 실습


<<interface>>
MyIterator
+getNext() : MyObject
+isEnd() : boolean
클라이언트
MyArrayIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayReverseIterator
+getNext() : MyObject
+isEnd() : boolean
MyConditionalIterator
+getNext() : MyObject
+isEnd() : boolean

Decorator 패턴 요약





예제 코드 분석

- decorator.u2
  - UnmodifiableList 클래스는 decorator 인가? proxy 인가?

- UnmodifiableList 클래스는,
  - 클라이언트에게 값을 리턴하는 read only 메소드만 MyList 객체에게 위임한다.
  - set, remove 등 수정 메소드는 MyList 객체에게 위임하지 않고 그냥 무시한다.

- 수정할 수 없는 read only 객체는 여러 스레드에 의해 공유되어도 문제가 발생하지 않는다.
  - 따라서, UnmodifiableList 객체를 proxy.e2 예제와 동일한 방법으로 multi thread 환경에서 실행해도 에러가 발생하지 않는다.
  - 목록이 변경되는 것을 원치 않으면서 MyList 객체를 전달할 때, UnmodifiableList가 유용할 수 있다

Abstract Factory

Abstract Factory

- 클라이언트에서 도구 클래스 이름을 명시하며 도구 객체를 생성하면 불필요한 의존성이 생긴다.

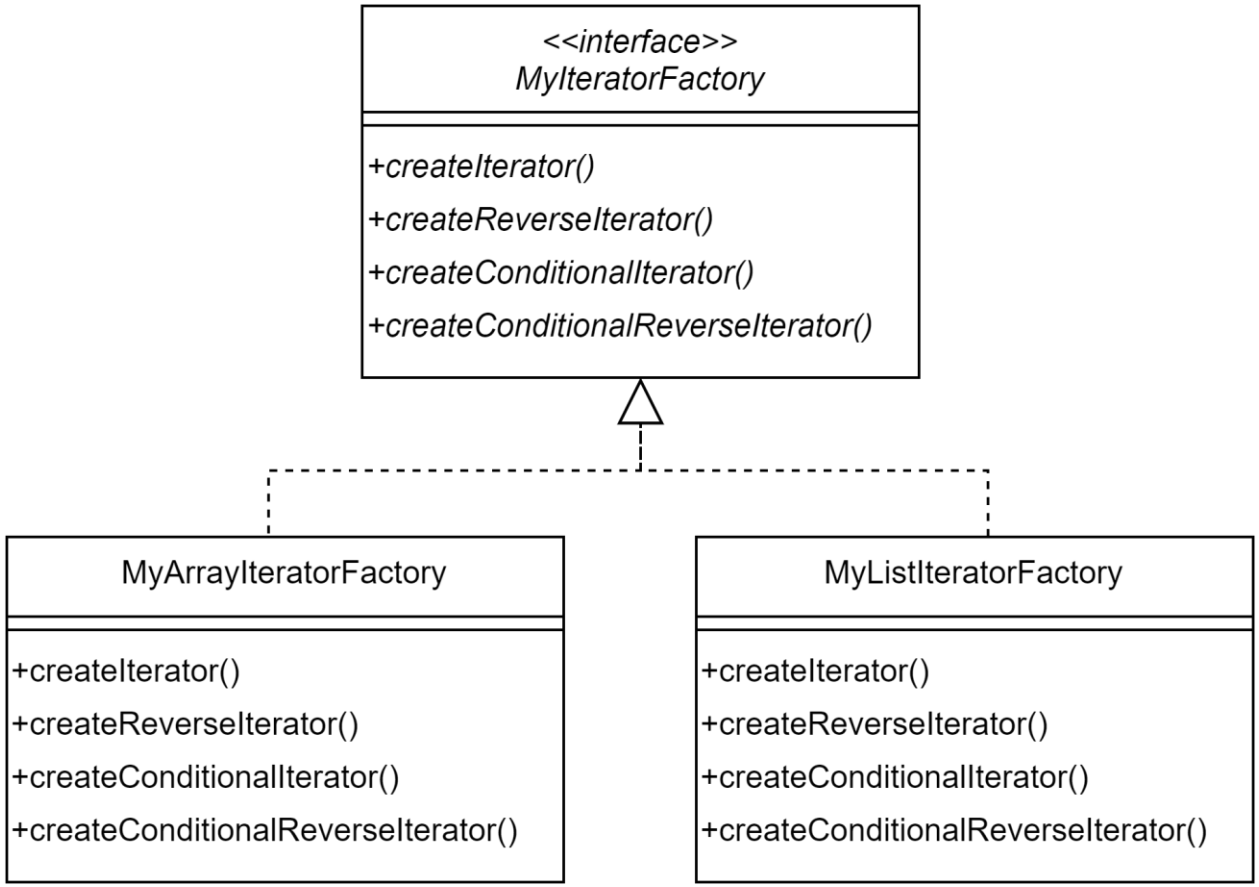
- 클라이언트에서 도구 클래스 이름을 명시하지 않고 도구 객체를 생성할 수 있으려면?

  → factory method 패턴을 구현한다

Abstract Factory

- 역할이 객체를 생성하는 것인 메소드 → factory method
- 역할이 객체를 생성하는 것인 클래스 → factory class
- factory method 와 factory method 패턴은 다르다
- factory method에 다형성을 구현하면 → factory method 패턴
- abstract factory
  - factory method 여러 개를 포함하는 factory class
  - 다형성이 구현된 factory class

Abstract Factory – 구조



예제 분석

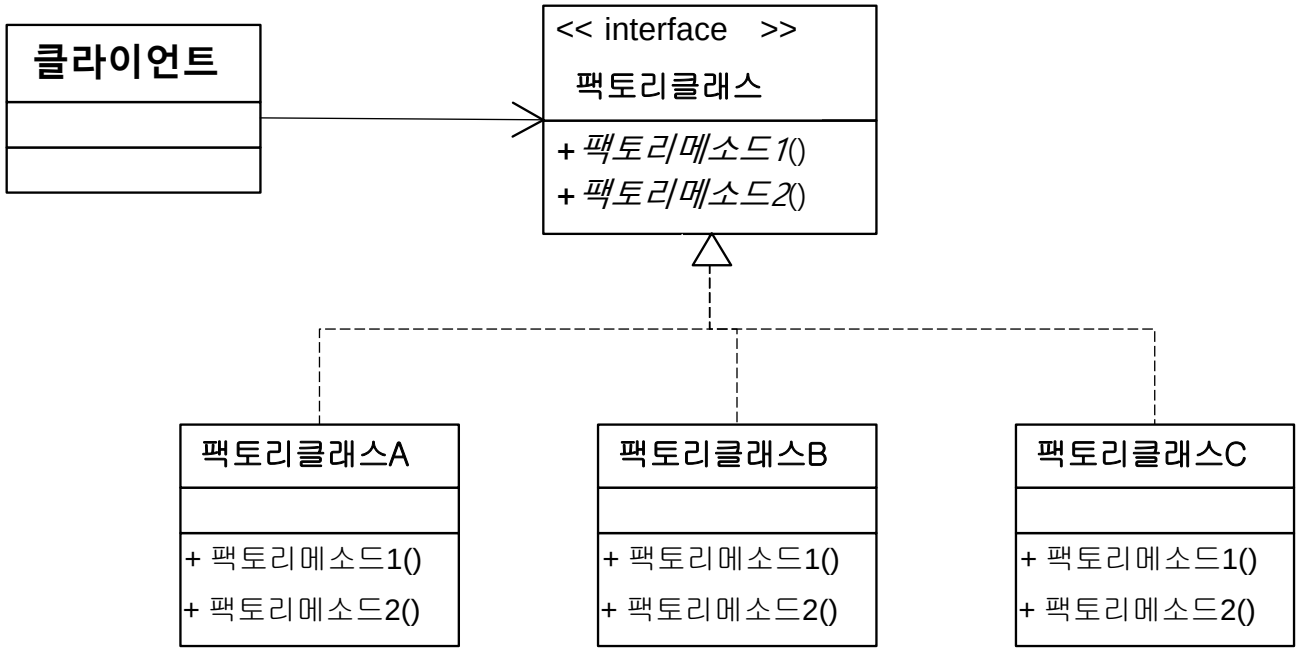
- abstract_factory.e1 예제
  - 앞 슬라이드의 abstract factory 구현함

구현 실습

- abstract_factory.e1 예제 pdf 파일 11p, 12p 녹색으로 칠한 부분은 유사한 코드 중복이다.

- 이 코드 중복을 어떻게 제거?

Abstract Factory 요약

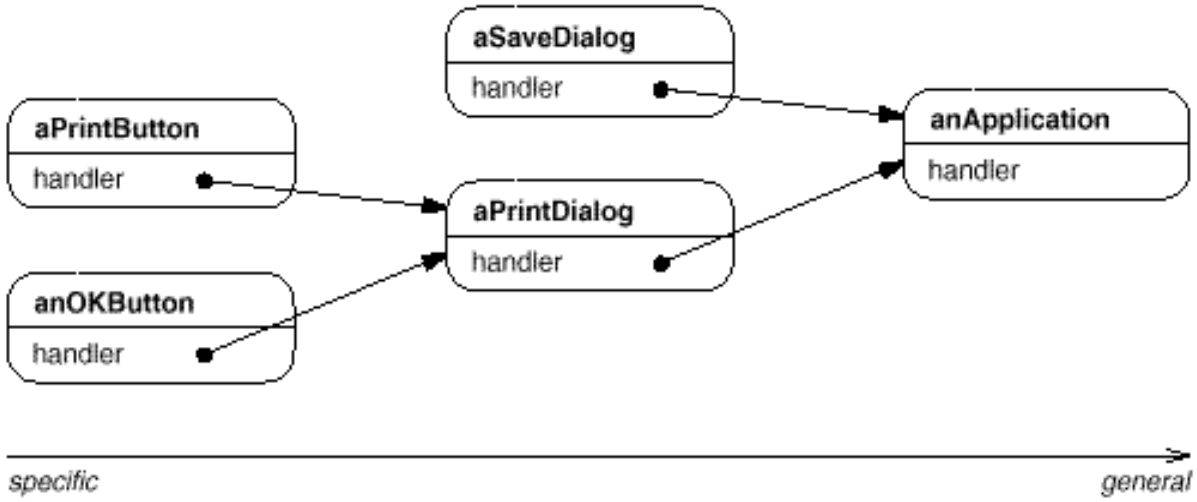


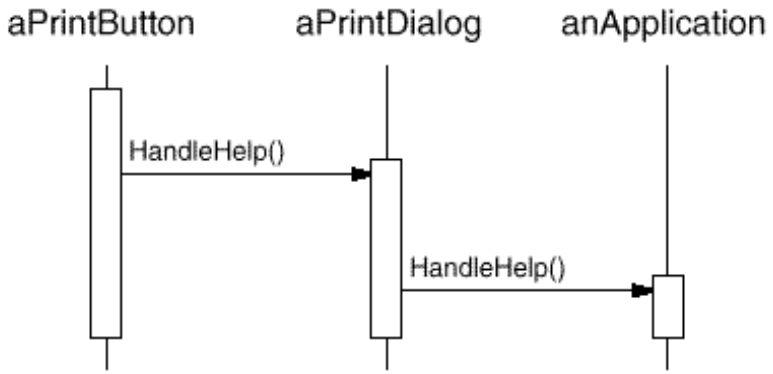
Chain of Responsibility

Chain of Responsibility – 의도

- GUI 어플리케이션에서 문맥 도움말 기능
- 사용자는 언제든지 F1 키를 눌러 도움말을 요청할 수 있다
- 현재 입력 포커스를 가지고 있는 컨트롤에 대한 도움말이 요청된다
- 만약 그 컨트롤에 대한 등록된 도움말이 없다면, 그 컨트롤을 포함하고 있는 panel 의 도움말이 요청된다
- panel 에도 등록된 도움말이 없다면, 그 panel 을 포함하고 있는 대화상자의 도움말이 요청된다

Chain of Responsibility – 의도





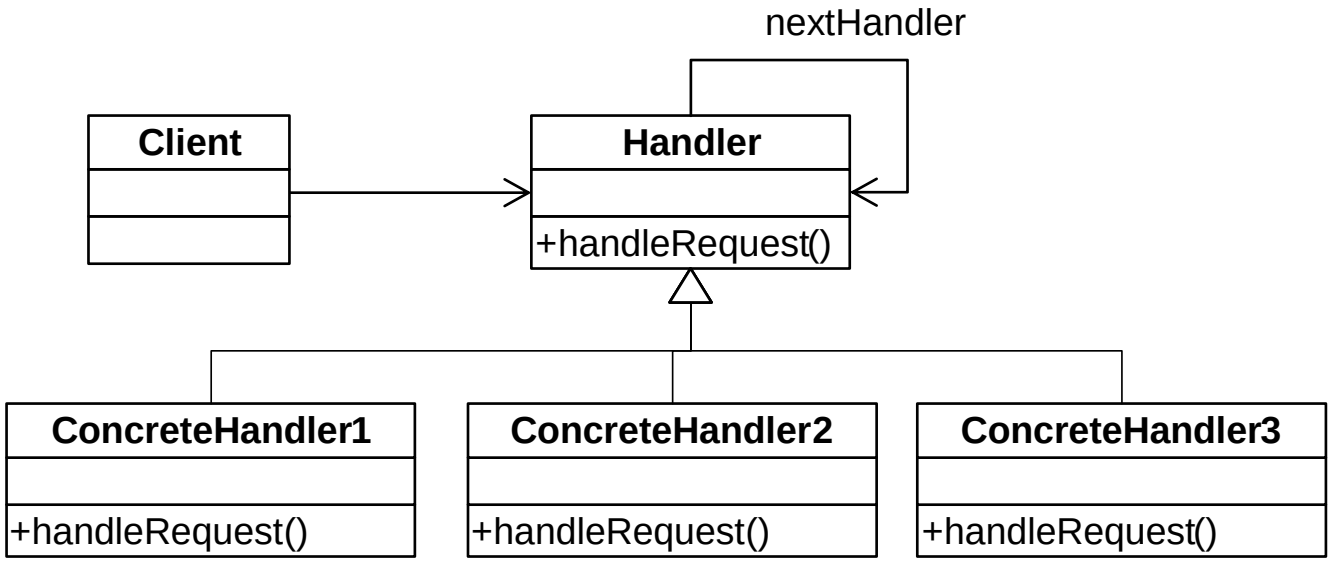
Chain of Responsibility – 의도

- 먼저 chain에서 첫 객체가 요청을 받는다.
  - 객체는 그것을 처리하거나
  - 아니면 chain에서 다음 객체에게 요청을 전달한다

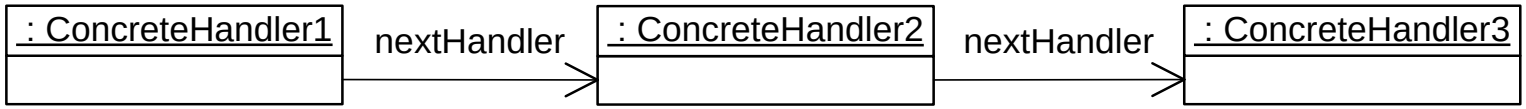
- 요청을 하는 쪽에서는 누가 그 요청을 받아서 처리할지 전혀 몰라도 된다

- 요청을 받아 처리하기 위해 chain에 참여할 객체들은 자유롭게 추가되고 제거될 수 있다

Chain of Responsibility – 구조



Chain of Responsibility – 구조



Chain of Responsibility – 결과

- 요청을 하는 쪽과 받는 쪽의 결합도가 매우 낮다
- 요청을 받는 객체가 자유롭게 추가 제거될 수 있다

예제 코드 분석

- chain_of_responsibility.e1
  - 전달된 명령을 어떤 객체가 처리하는가?
  - View1, View2, Document, Window, MainWindow, Application

- chain_of_responsibility.e2
  - 전달된 로그 메시지는 어떤 Logger 객체가 처리하는가?

State

State – 의도

- 상태 다이어그램을 구현하면 보통 거대한 switch 문이 많이 만들어진다.
  - 그 클래스에 전달되는 이벤트는 메소드로 구현됨
  - 이벤트를 구현한 메소드에 switch 문이 만들어짐
  - switch 문의 case 는 각 상태값

- 상태 다이어그램을 구현한 switch 문이 너무 크면, 코드 가독성이 떨어지고 유지보수가 어렵다.

- switch문 없이 복잡한 상태 다이어그램을 구현하는 방법은?

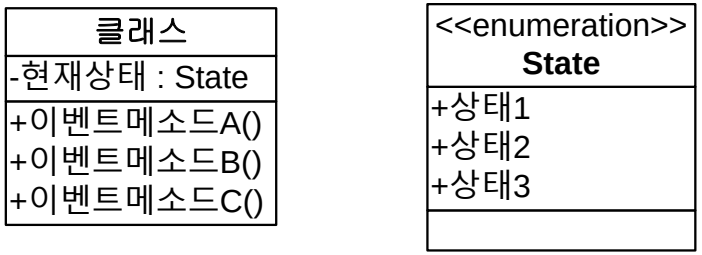
State – 구현방법1

```
클래스 {
    State 현재상태;

    이벤트메소드A() {
       switch(현재상태) {
          case 상태1: 작업A1;
          case 상태2: 작업A2;
          case 상태3: 작업A3;
        }
    }
    이벤트메소드B() {
        switch(현재상태) {
          case 상태1: 작업B1;
          case 상태2: 작업B2;
          case 상태3: 작업B3;
        }
    }
    이벤트메소드C() {
        switch(현재상태) {
           case 상태1: 작업C1;
           case 상태2: 작업C2;
           case 상태3: 작업C3;
        }
    }
}
```

상태의 전이는
현재상태 멤버변수값의 변경으로 구현됨

State – 구현방법1


클래스
-현재상태 : State
+이벤트메소드A()
+이벤트메소드B()
+이벤트메소드C()
<<enumeration>>
State
+상태1
+상태2
+상태3

State – 구현방법2

- 각 상태를 내부 클래스로 추출하여 구현한다.

- 현재상태 멤버변수는 현재 태에 해당하는 상태 객체를 가르킨다

State – 구현방법2

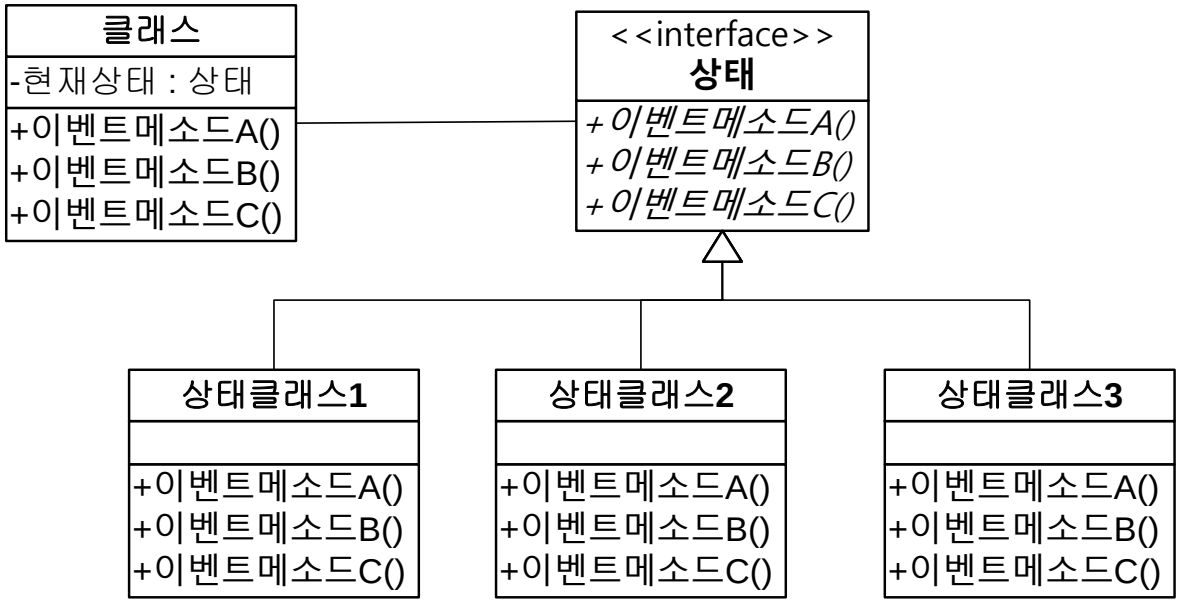
```
상태클래스1 extends 상태 {
    이벤트메소드A() { 작업A1; }
    이벤트메소드B() { 작업B1; }
    이벤트메소드C() { 작업C1; }
}
상태클래스2 extends 상태 {
    이벤트메소드A() { 작업A2; }
    이벤트메소드B() { 작업B2; }
    이벤트메소드C() { 작업C2; }
}
상태클래스3 extends 상태 {
    이벤트메소드A() { 작업A3; }
    이벤트메소드B() { 작업B3; }
    이벤트메소드C() { 작업C3; }
}

클래스 {
    상태 현재상태;

    이벤트메소드A() { 현재상태객체.이벤트메소드A(); }
    이벤트메소드B() { 현재상태객체.이벤트메소드B(); }
    이벤트메소드C() { 현재상태객체.이벤트메소드C(); }
}
```

상태의 전이는
멤버변수 현재상태가 가르키는
상태객체의 변경으로 구현됨

State – 구현방법2


클래스
-현재상태 : 상태
+이벤트메소드A()
+이벤트메소드B()
+이벤트메소드C()
<<interface>>
상태
+이벤트메소드A()
+이벤트메소드B()
+이벤트메소드C()
상태클래스1
+이벤트메소드A()
+이벤트메소드B()
+이벤트메소드C()
상태클래스2
+이벤트메소드A()
+이벤트메소드B()
+이벤트메소드C()
상태클래스3
+이벤트메소드A()
+이벤트메소드B()
+이벤트메소드C()

State – 결과

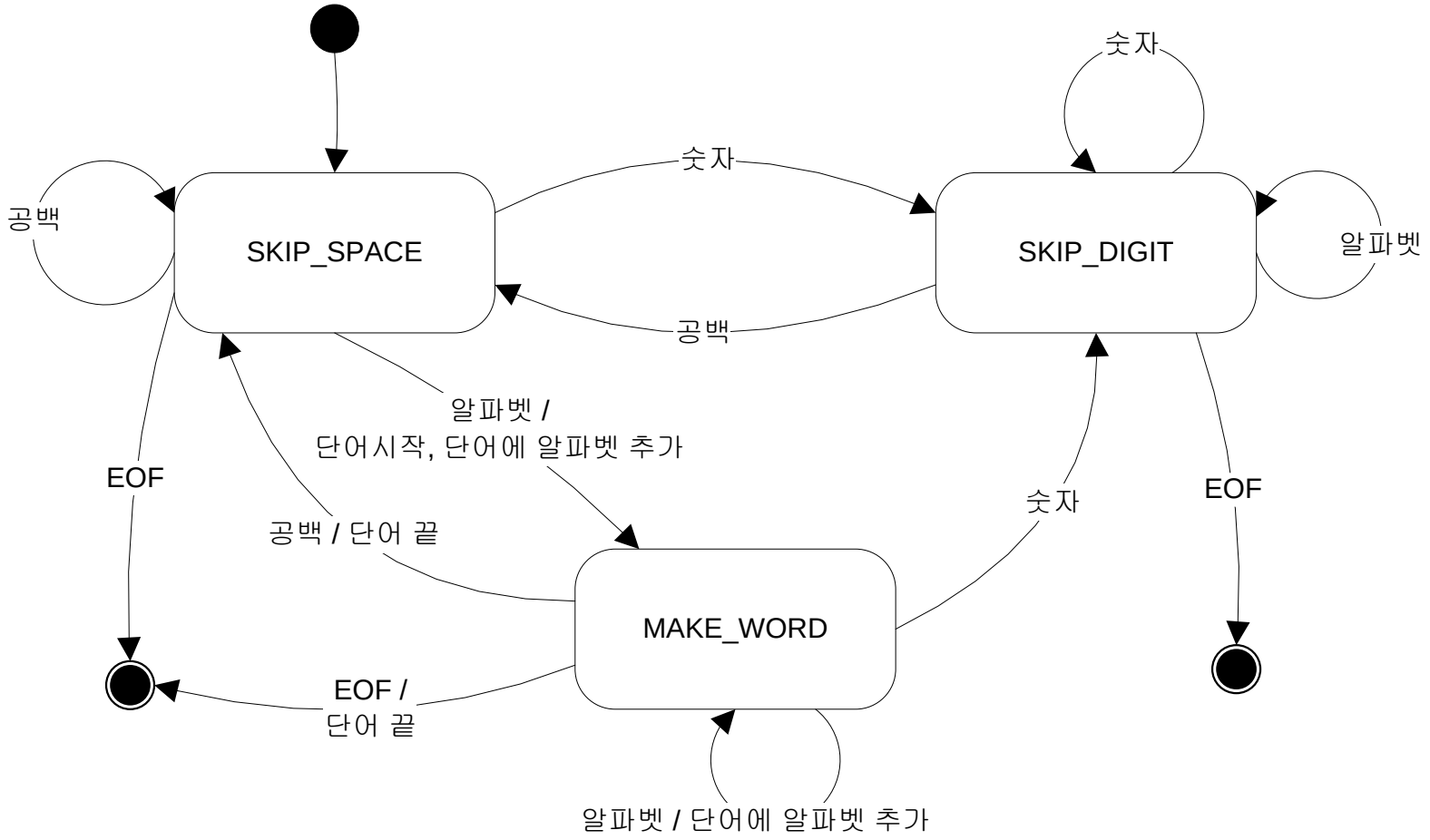
- 각 상태의 코드가 switch 문의 case 로 분산되지 않고 하나의 클래스에 모여있기 때문에 가독성이 향상된다
- 상태의 추가가 쉽다
- 코드의 양은 약간 증가한다

단어 수 세기

- 파일에서 단어 수 세는 프로그램 구현

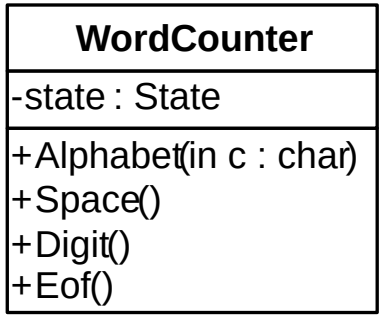
- 알파벳만 모인 것이 단어
- 숫자와 인접해 있는 단어는 단어로 인정하지 않는다
- 알파벳도 아니고 숫자도 아닌 문자는 공백으로 취급

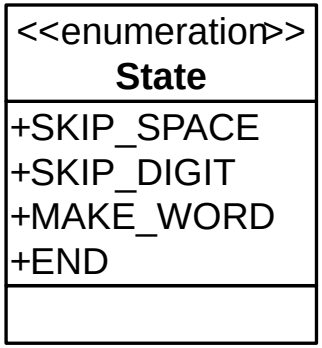
단어 수 세기


숫자
숫자
공백
SKIP_SPACE
SKIP_DIGIT
알파벳
공백
알파벳 /
단어시작, 단어에 알파벳 추가
EOF
숫자
EOF
공백 / 단어 끝
MAKE_WORD
EOF /
단어 끝
알파벳 / 단어에 알파벳 추가

단어 수 세기

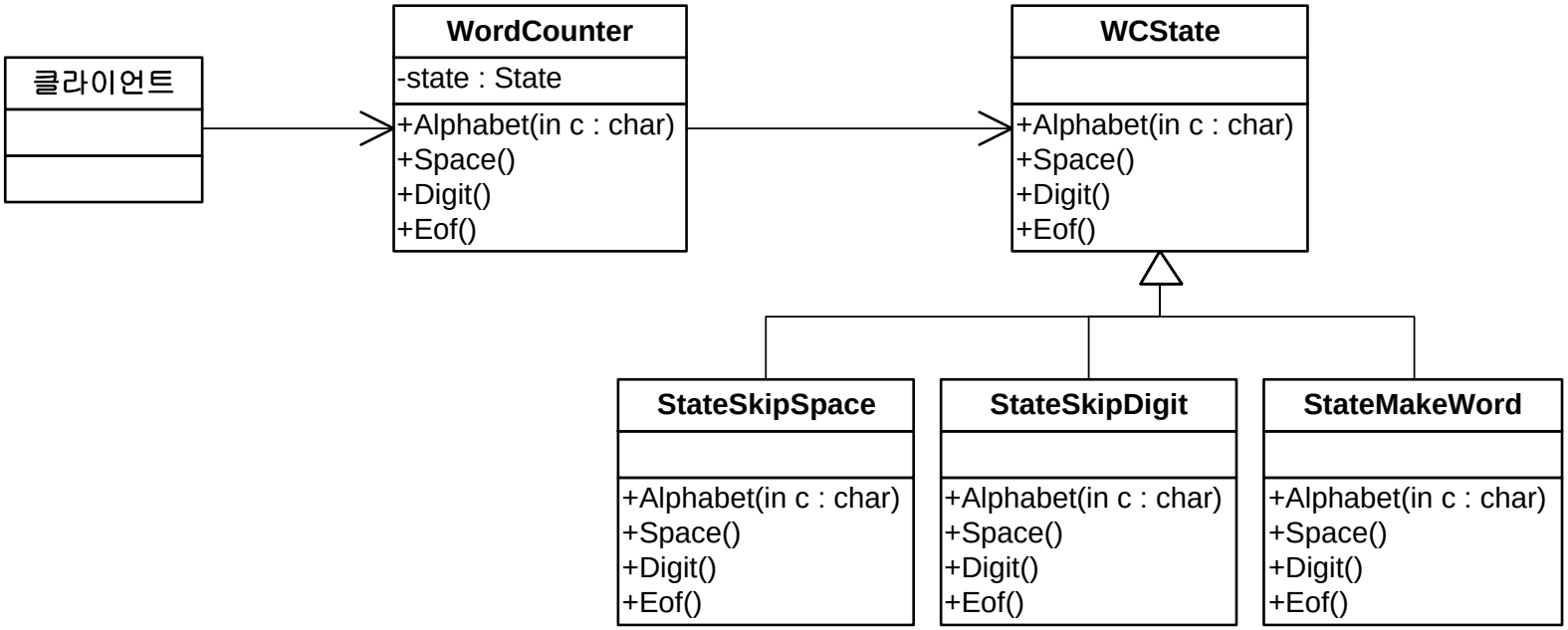
- 클래스 하나로 구현하기
  - 이벤트 → 메소드
  - 상태 → 멤버 변수





단어 수 세기

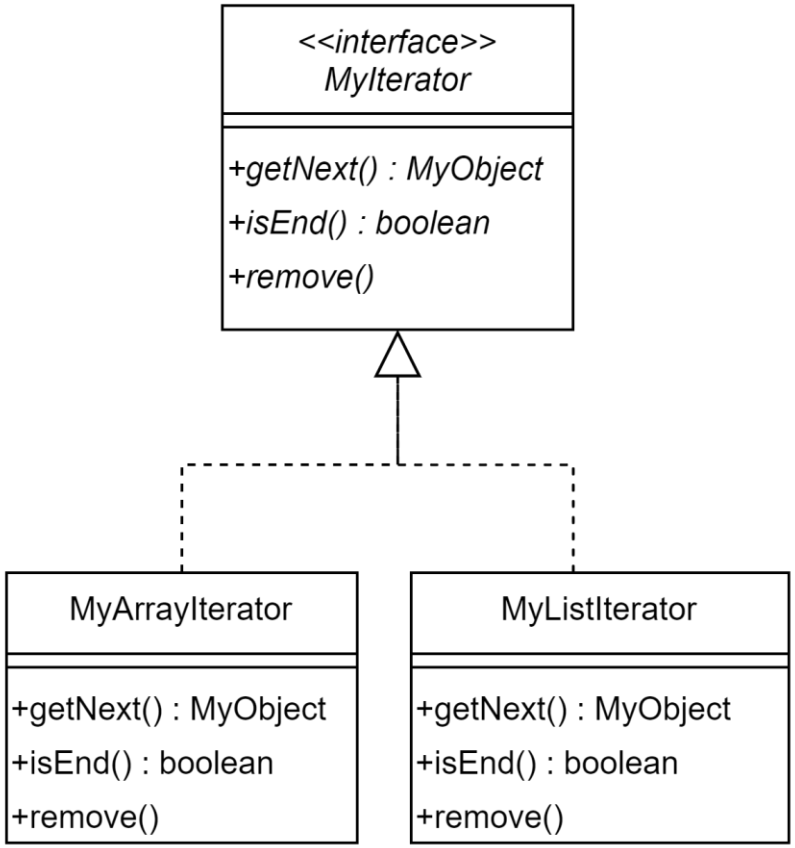
- State 패턴으로 구현하기



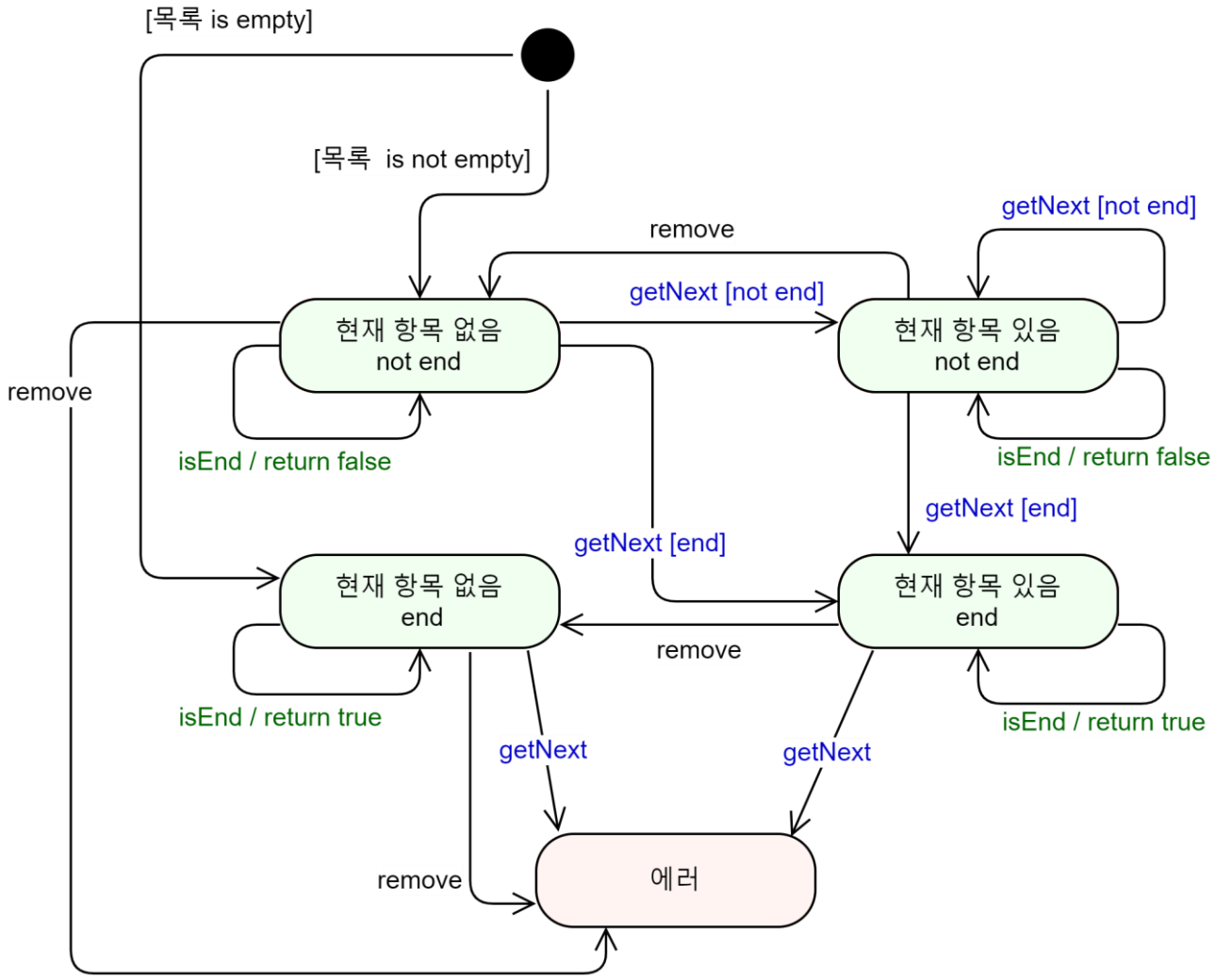
예제 코드 분석

- state.e1 → switch 문으로 statechart 구현
- state.e2 → example1에 state 패턴을 적용
- state.e3 → 이벤트 위주 로직이 아니고 절차적 로직으로 구현

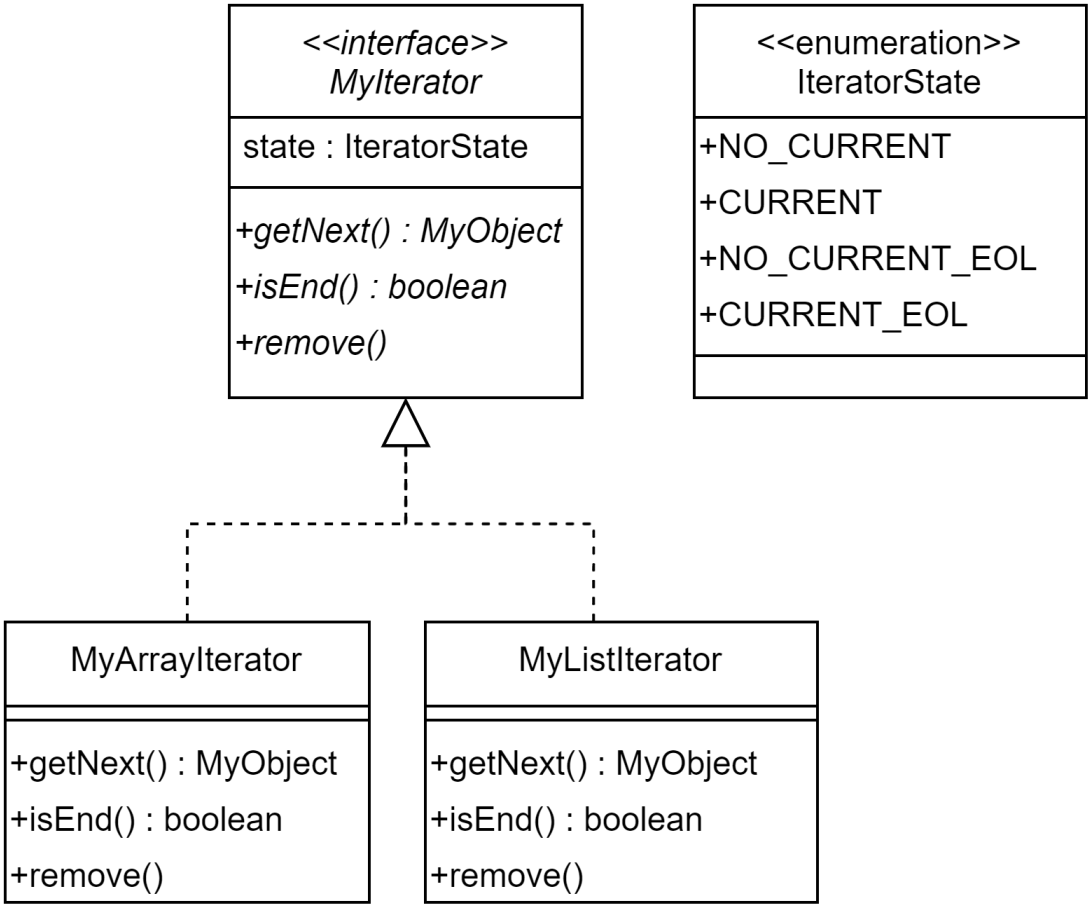
iterator


<<interface>>
MyIterator
+getNext() : MyObject
+isEnd() : boolean
+remove()
MyArrayIterator
+getNext() : MyObject
+isEnd() : boolean
+remove()
MyListIterator
+getNext() : MyObject
+isEnd() : boolean
+remove()

iterator


[목록 is empty]
[목록 is not empty]
remove
getNext [not end]
getNext [not end]
현재 항목 없음
not end
현재 항목 있음
not end
remove
isEnd / return false
isEnd / return false
getNext [end]
getNext [end]
현재 항목 없음
end
현재 항목 있음
end
remove
isEnd / return true
isEnd / return true
getNext
getNext
remove
에러

iterator

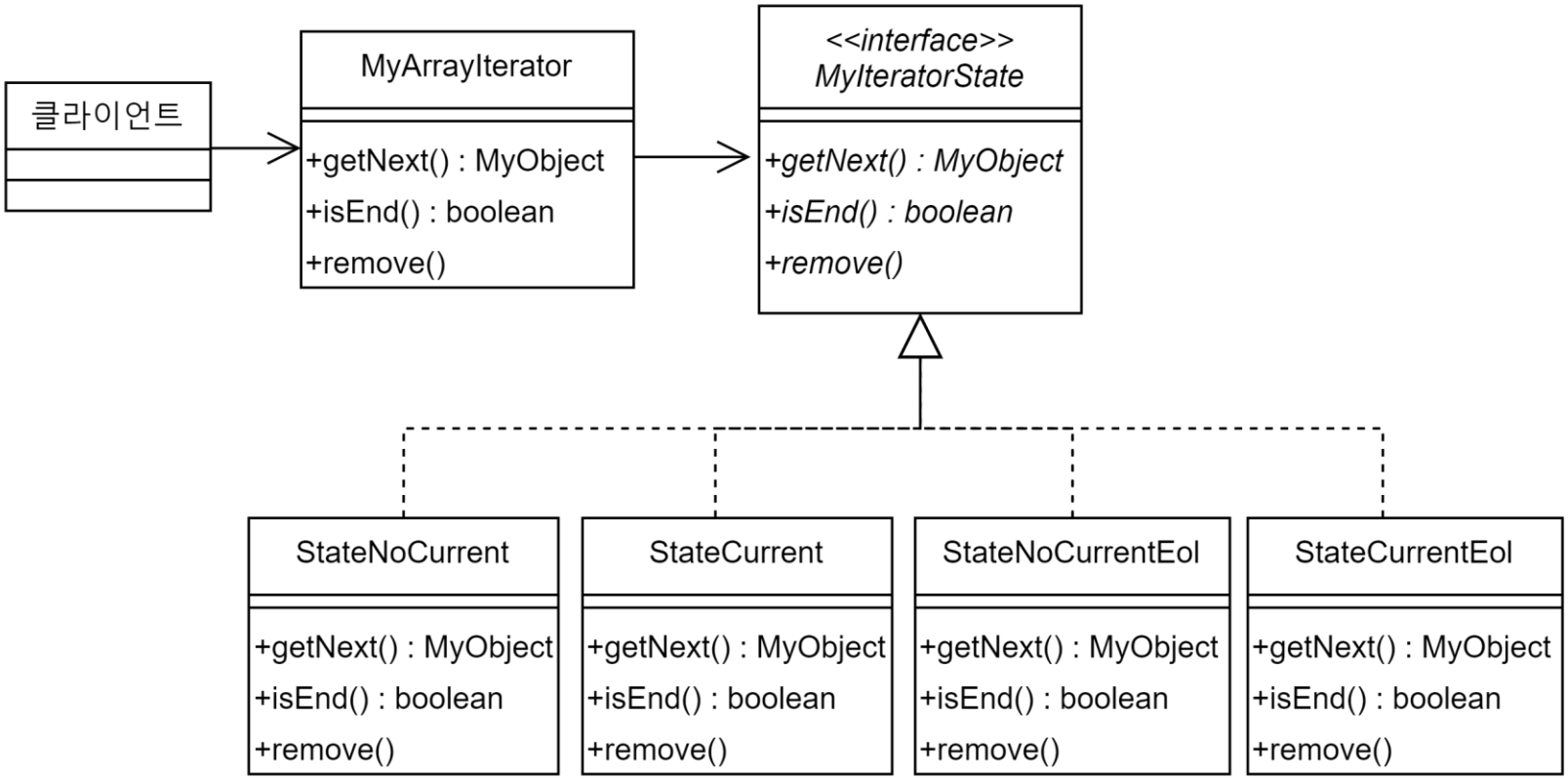

<<interface>>
MyIterator
state : IteratorState
+getNext() : MyObject
+isEnd() : boolean
+remove()
<<enumeration>>
IteratorState
+NO_CURRENT
+CURRENT
+NO_CURRENT_EOL
+CURRENT_EOL
MyArrayIterator
+getNext() : MyObject
+isEnd() : boolean
+remove()
MyListIterator
+getNext() : MyObject
+isEnd() : boolean
+remove()

예제 코드 분석

- state.e4 MyArrayIterator
  - 모든 상태 꼼꼼히 고려하지 않은, 안전하지 않은 구현

- state.e5 MyArrayIterator
  - 모든 상태를 꼼꼼히 고려한, 안전한 구현
  - 앞 슬라이드 구조 구현

state 패턴


클라이언트
MyArrayIterator
+getNext() : MyObject
+isEnd() : boolean
+remove()
<<interface>>
MyIteratorState
+getNext() : MyObject
+isEnd() : boolean
+remove()
StateNoCurrent
+getNext() : MyObject
+isEnd() : boolean
+remove()
StateCurrent
+getNext() : MyObject
+isEnd() : boolean
+remove()
StateNoCurrentEol
+getNext() : MyObject
+isEnd() : boolean
+remove()
StateCurrentEol
+getNext() : MyObject
+isEnd() : boolean
+remove()

예제 코드 분석

- state.e6
  - state 패턴 구현
  - 앞 슬라이드 구조 구현

예제 코드 분석

- state.e6 구현에는 코드 중복이 있다.
  - pdf 파일에서 녹색으로 칠한 부분이 코드 중복
  - 클래스 하나에 있었던 코드를
    여러 상태 클래스로 쪼개었기 때문에 발생한 코드 중복

- 이 코드 중복을 제거하는 방법은? → state.e7

Bridge

Bridge - 문제

- high level operation 과 low level operation 의 분리 구현

- 그래픽 에디터에서 high level operation
  - 도형 좌표 이동
  - 도형 그리기
  - 도형 이동 중 임시로 그리기
  - 도형의 그림자 그리기

- 그래픽 에디터에서 low level operation
  - Android 창에 직선 그리기, 사각형 그리기, 색칠하기
  - AWT 창에 직선 그리기, 사각형 그리기, 색칠하기

Bridge - 문제

- 안드로이드나 AWT에서 그래픽 에디터의 도형 클래스의 구현은 서로 꽤 비슷할 것이다.
  - 좌표 이동, 회전의 구현 등

- 하지만 플랫폼 API가 다르기 때문에 약간씩 달라질 수 밖에 없다.
  - 도형 그리기, 그림자 그리기 등

Bridge - 문제

- 도형 클래스를 각각의 플랫폼 마다 따로 구현해야 하나?

- Android Rectangle, Android Line, Android Text ...
- AWT Rectangle, Windows Line, Windows Text ...
- SWT Rectangle, Windows Line, Windows Text ...

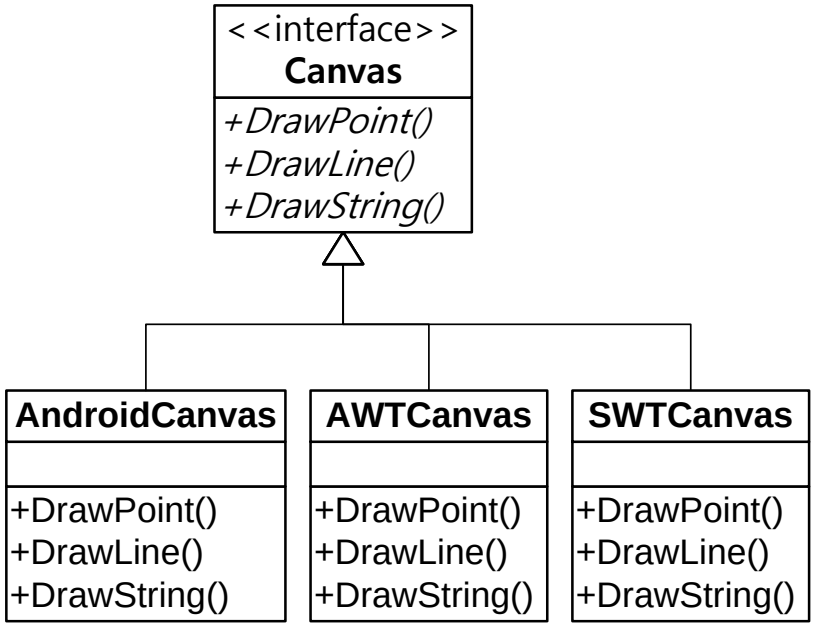
- 구현할 도형 클래스의 수 = 플랫폼의 수 x 도형의 수

- 플랫폼에 무관하게 도형 클래스를 구현할 수는 없을까?

- 참고: Google의 Android, Sun의 AWT, IBM의 SWT

Bridge – 해결

플랫폼 API 호출 대행 클래스

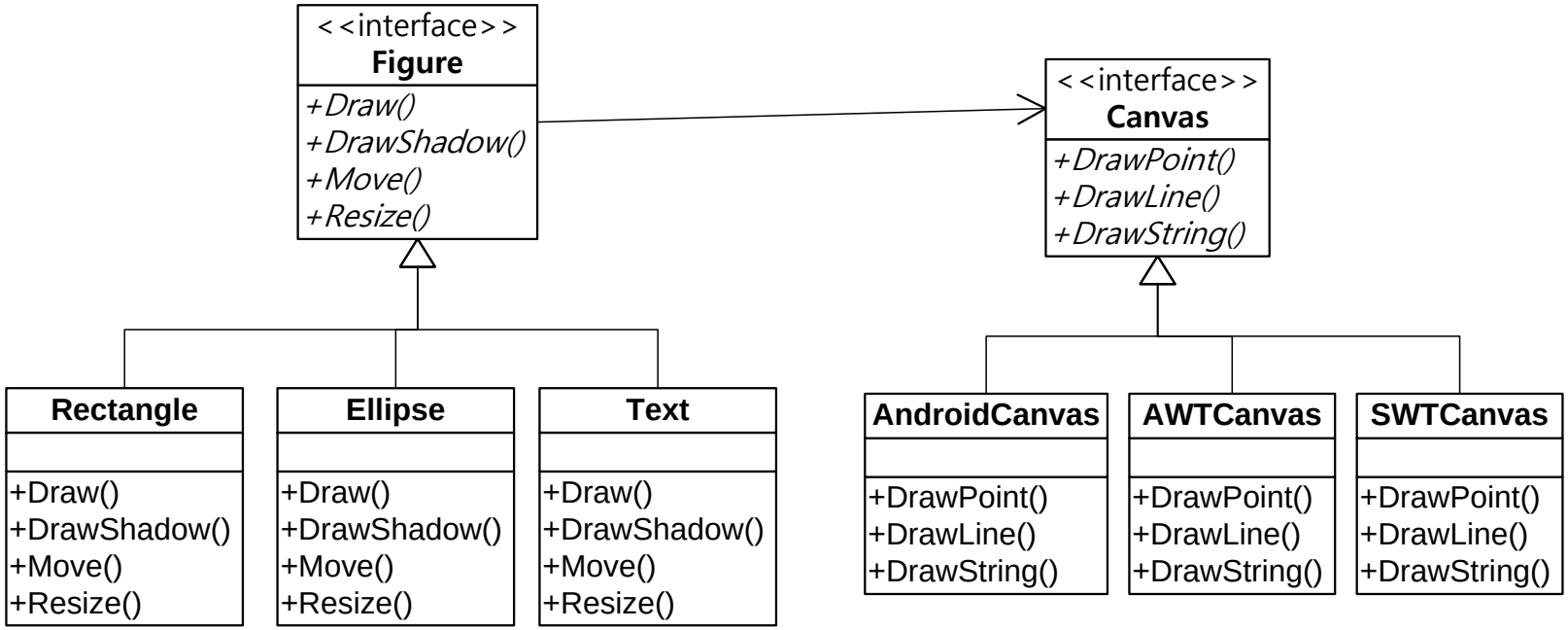


Bridge – 해결

- 도형 클래스는 직접 플랫폼 API를 호출하지 않고, Canvas 객체의 메소드를 호출하여 화면에 출력한다

- Canvas에 다형성이 구현되어 있으므로, 도형 클래스는 Canvas의 자식 클래스들을 참조할 필요 없이, Canvas 인터페이스만 참조하여 구현될 수 있다.

Bridge – 해결

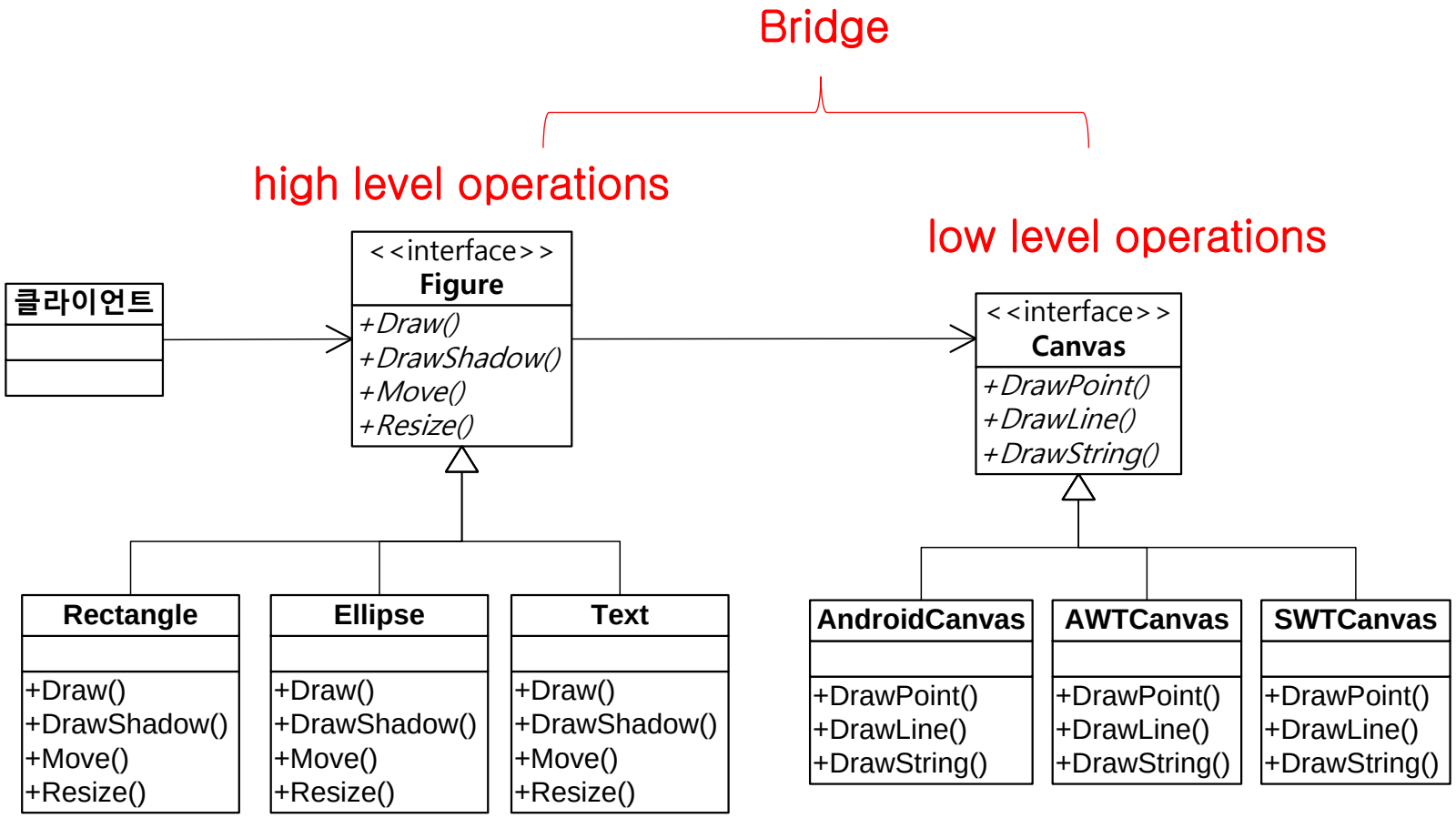

<<interface>>
Figure
+Draw()
+DrawShadow()
+Move()
+Resize()
<<interface>>
Canvas
+DrawPoint()
+DrawLine()
+DrawString()
Rectangle
+Draw()
+DrawShadow()
+Move()
+Resize()
Ellipse
+Draw()
+DrawShadow()
+Move()
+Resize()
Text
+Draw()
+DrawShadow()
+Move()
+Resize()
AndroidCanvas
+DrawPoint()
+DrawLine()
+DrawString()
AWTCanvas
+DrawPoint()
+DrawLine()
+DrawString()
SWTCanvas
+DrawPoint()
+DrawLine()
+DrawString()

Bridge – 구조

- 클라이언트인 그래픽 에디터 윈도우는 도형을 화면에 그릴 때, Figure 인터페이스의 메소드를 호출한다.
  - draw(), DrawShadow()
  - high level operations

- 도형은 Canvas 인터페이스의 메소드를 호출하여 화면에 그린다
  - DrawPoint(), drawLine(), ...
  - low level operations

Bridge – 구조


Bridge
high level operations
low level operations
클라이언트
<<interface>>
Figure
+Draw()
+DrawShadow()
+Move()
+Resize()
<<interface>>
Canvas
+DrawPoint()
+DrawLine()
+DrawString()
Rectangle
+Draw()
+DrawShadow()
+Move()
+Resize()
Ellipse
+Draw()
+DrawShadow()
+Move()
+Resize()
Text
+Draw()
+DrawShadow()
+Move()
+Resize()
AndroidCanvas
+DrawPoint()
+DrawLine()
+DrawString()
AWTCanvas
+DrawPoint()
+DrawLine()
+DrawString()
SWTCanvas
+DrawPoint()
+DrawLine()
+DrawString()

Bridge – 결과

- 클라이언트인 그래픽 에디터는 Figure 인터페이스만 참조해서 구현될 수 있다.
- 클라이언트는 특정 플랫폼과 무관하게 구현될 수 있다.
- 새 도형 클래스가 추가될 때, 클라이언트와 Canvas 는 영향 받지 않는다.
- 새 플랫폼을 지원하기 위해 새 Canvas 클래스가 추가될 때, 클라이언트와 도형은 영향을 받지 않는다.

Bridge - 문제

- high level operation 과 low level operation 의 분리 구현

- 사진 장식 어플리케이션의 예

- high level operation : 장식효과
  - 사진에 경계선 효과 넣기
  - 사진에 copyright 텍스트 넣기
  - 사진에 말 풍선 넣기

- low level operation : 사진 파일 다루기
  - jpeg 그림에 선 그리기
  - gif 그림에 문자 출력하기
  - bmp 파일 읽어오기/저장하기

Bridge - 문제

- 사진에 말 풍선을 넣기 장식 효과의 절차
  - 사진 파일을 읽는다
  - 말 풍선의 선을 그린다
  - 말 풍선 내부를 칠한다
  - 텍스트를 출력한다
  - 사진 파일을 저장한다

- 세부 스텝 구현은 사진 파일의 종류에 따라 다르다
  - jpeg, gif, bmp ...
  - (사실 다르지 않지만 다르다고 가정하고..)

Bridge - 문제

- 말 풍선 넣기 같은 장식 효과를 구현할 때 사진 파일의 종류 마다 따로 구현해야 하나?

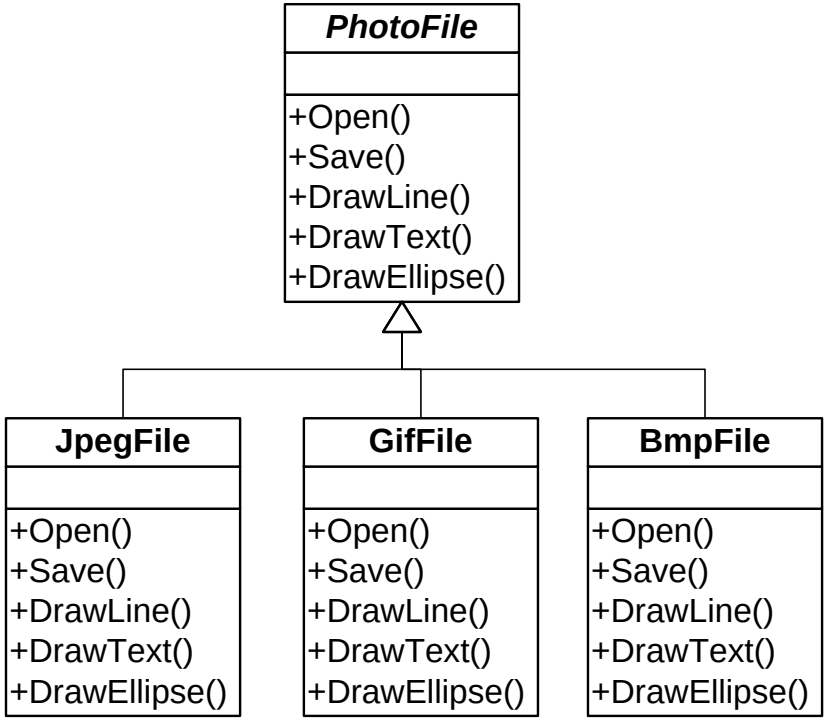
- jpeg에 말 풍선 넣기 메소드, gif에 말 풍선 넣기 메소드...
  jpeg에 경계선 효과 메소드, gif에 경계선 효과 메소드...

- 구현할 메소드의 수 = 장식 효과 종류의 수 x 사진 파일의 종류 수

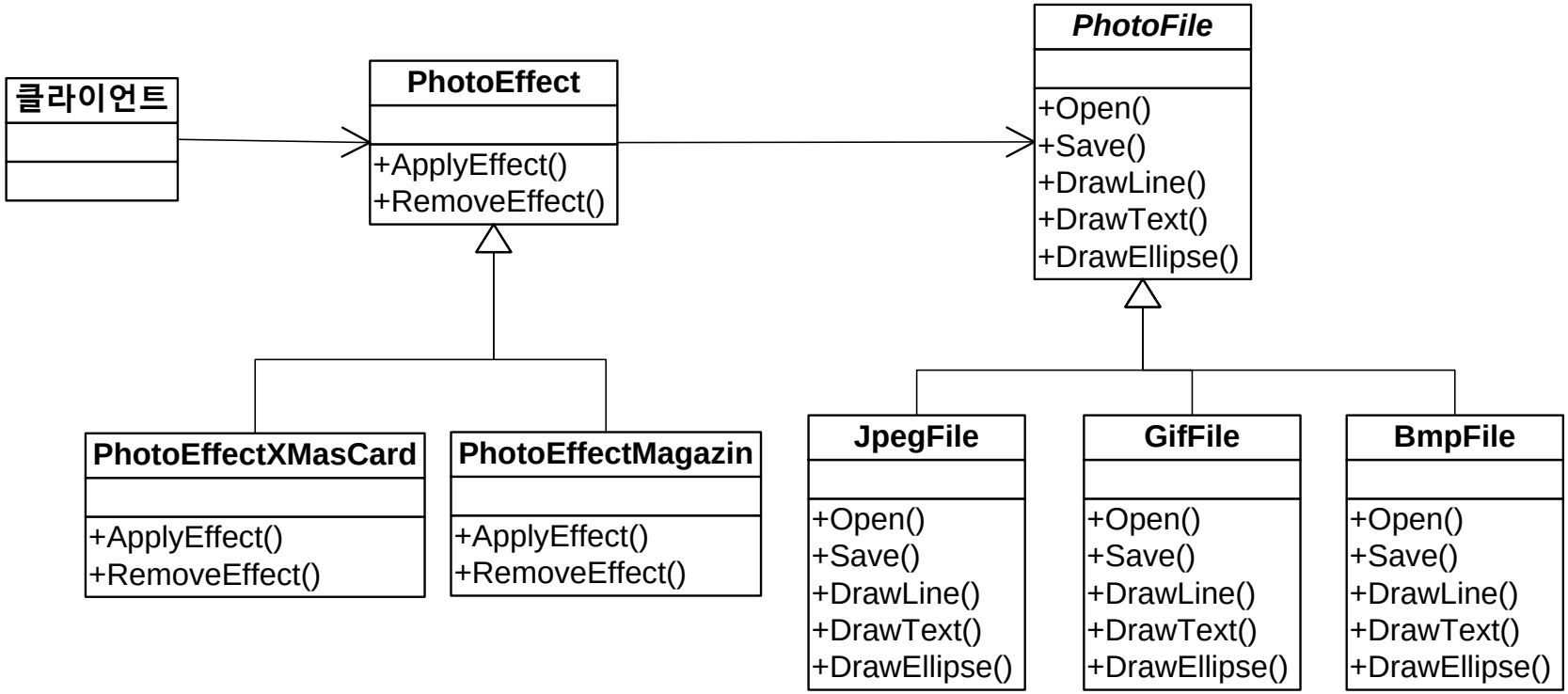
- 사진 파일과 무관하게 장식 효과를 구현할 수는 없나?

Bridge – 구조

사진 파일 구현 클래스



Bridge – 구조


클라이언트
PhotoEffect
+ApplyEffect()
+RemoveEffect()
PhotoFile
+Open()
+Save()
+DrawLine()
+DrawText()
+DrawEllipse()
PhotoEffectXMasCard
+ApplyEffect()
+RemoveEffect()
PhotoEffectMagazin
+ApplyEffect()
+RemoveEffect()
JpegFile
+Open()
+Save()
+DrawLine()
+DrawText()
+DrawEllipse()
GifFile
+Open()
+Save()
+DrawLine()
+DrawText()
+DrawEllipse()
BmpFile
+Open()
+Save()
+DrawLine()
+DrawText()
+DrawEllipse()

예제 코드 분석

- bridge.e1

- 왼쪽의 장식 효과 객체 하나와 오늘쪽의 사진 조작 구현 객체 중 하나를 선택하여 두 객체를 결합하면 원하는 형식의 사진 파일에 원하는 장식 효과를 줄 수 있다

- 구현할 클래스의 수 = 장식 효과 종류의 수 + 사진 파일의 종류 수

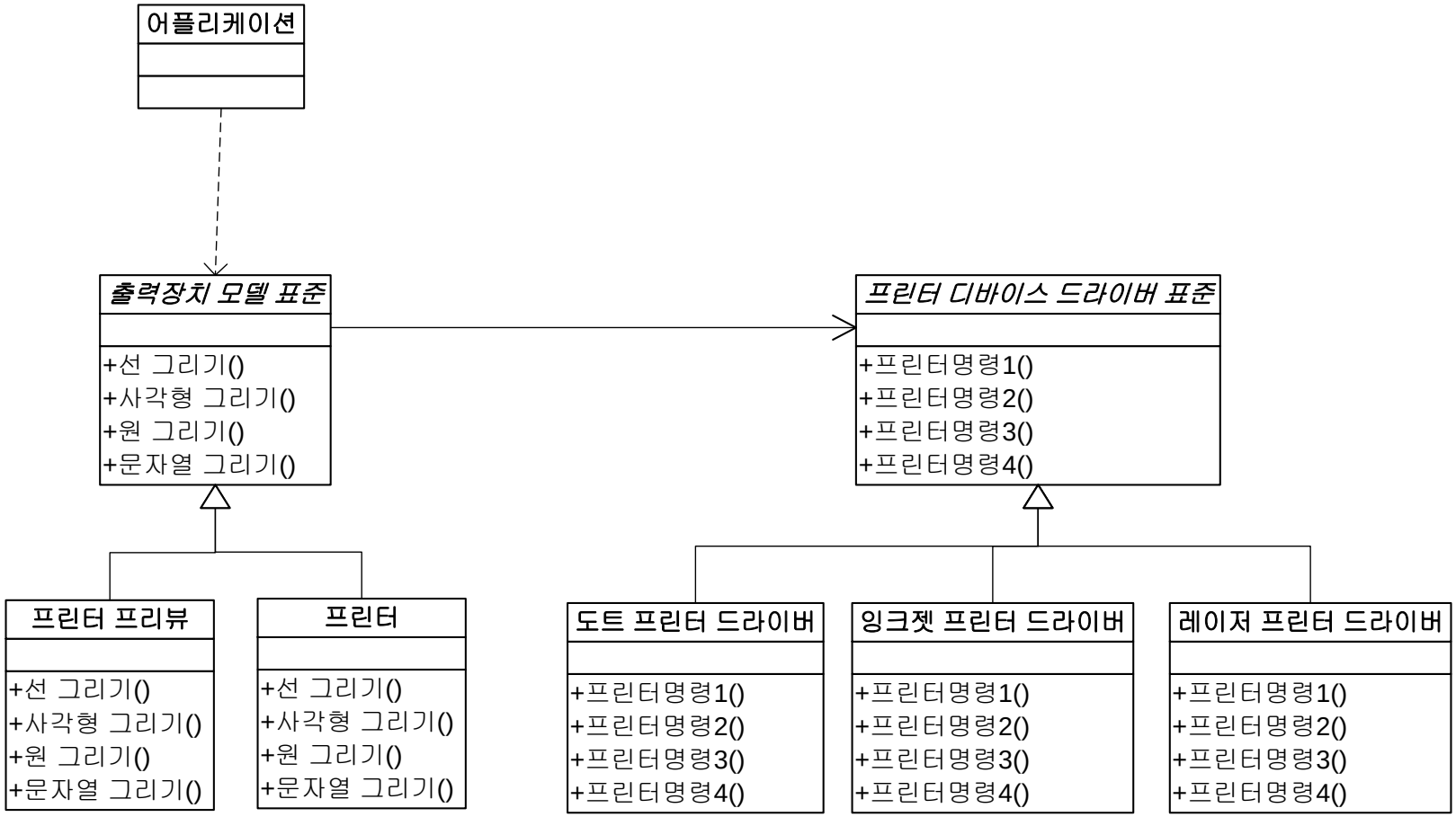
Bridge – 결과

- 클라이언트인 사진 편집기는 장식효과의 종류와 사진파일의 종류에 무관하게 구현될 수 있다.

- 지원할 사진 파일의 종류가 추가될 때 사진 파일 클래스 하나만 추가하면 된다

- 장식 효과를 추가할 때 장식 효과 클래스 하나만 추가하면 된다

Bridge 패턴의 예


어플리케이션
출력장치 모델 표준
+선 그리기()
+사각형 그리기()
+원 그리기()
+문자열 그리기()
프린터 디바이스 드라이버 표준
+프린터명령1()
+프린터명령2()
+프린터명령3()
+프린터명령4()
프린터 프리뷰
+선 그리기()
+사각형 그리기()
+원 그리기()
+문자열 그리기()
프린터
+선 그리기()
+사각형 그리기()
+원 그리기()
+문자열 그리기()
도트 프린터 드라이버
+프린터명령1()
+프린터명령2()
+프린터명령3()
+프린터명령4()
잉크젯 프린터 드라이버
+프린터명령1()
+프린터명령2()
+프린터명령3()
+프린터명령4()
레이저 프린터 드라이버
+프린터명령1()
+프린터명령2()
+프린터명령3()
+프린터명령4()

Bridge 패턴의 예

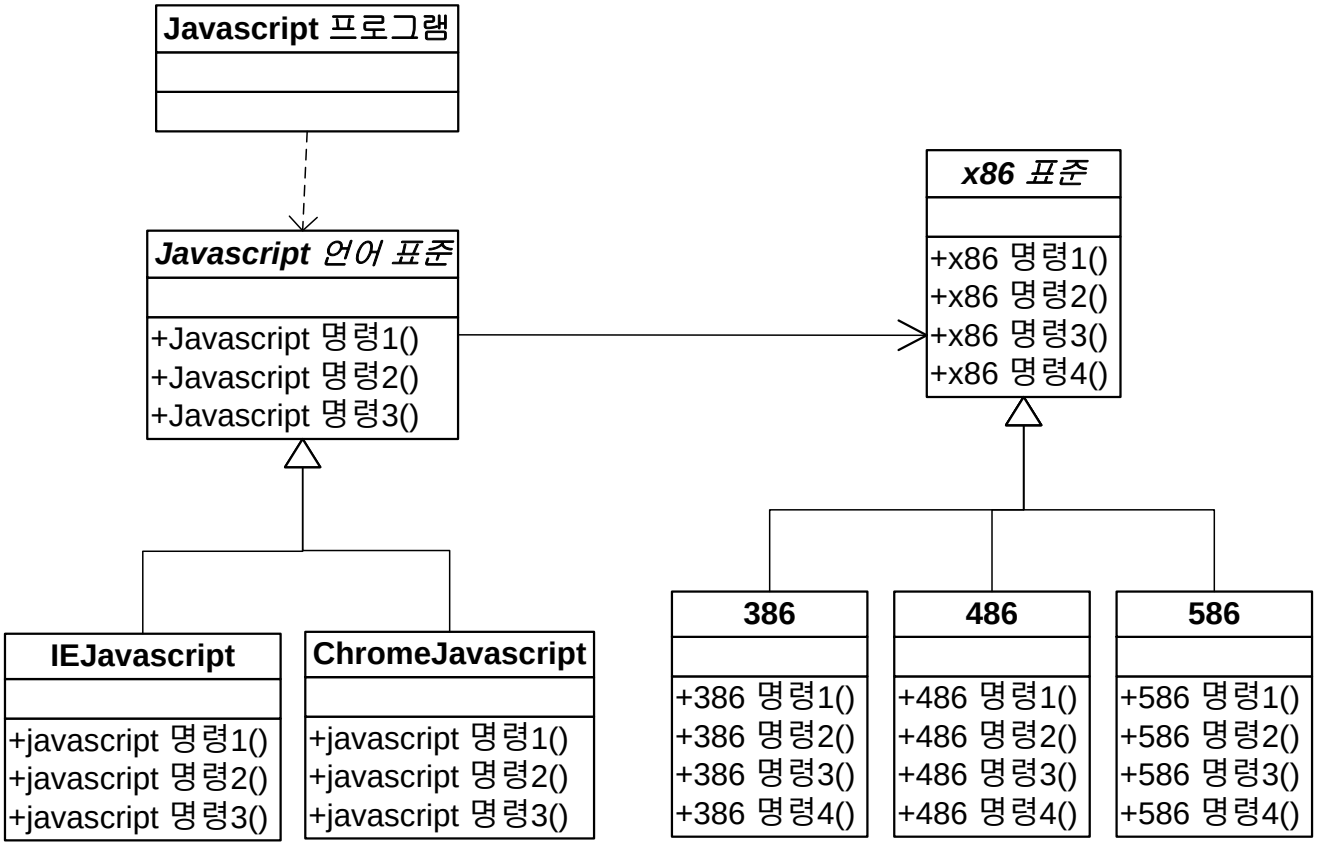
- 도트 프린터와 레이저 프린터는 명령 세트가 크게 다르다
- 어플리케이션이 다양한 프린터를 직접 다루는 것은 불가능하다
- 프린터들의 표준이 필요하다
  - 프린터 표준은 다양한 프린터들의 표준을 정의하는 것이 목적
  - 프린터 표준의 명령들은 매우 복잡할 수 밖에 없다
  - 어플리케이션에서 직접 호출하기에
    프린터 표준 명령들은 너무 낮은 단계의 명령
- 어플리케이션에서 사용하기 편한 높은 단계의 출력장치 표준이 필요

Bridge 패턴의 예

- 비유를 하자면
  - 프린터 디바이스 드라이버 표준 명령 == 어셈블리어 명령
  - 출력장치 모델 표준 명령 == 고급 언어

- 어플리케이션은 출력장치 모델 표준의 명령을 사용하여 구현됨
  - 이 명령은 실행시에 프린터 표준 명령으로 번역되어 실행된다

Bridge 패턴의 예


Javascript 프로그램
Javascript 언어 표준
+Javascript 명령1()
+Javascript 명령2()
+Javascript 명령3()
x86 표준
+x86 명령1()
+x86 명령2()
+x86 명령3()
+x86 명령4()
IEJavascript
+javascript 명령1()
+javascript 명령2()
+javascript 명령3()
ChromeJavascript
+javascript 명령1()
+javascript 명령2()
+javascript 명령3()
386
+386 명령1()
+386 명령2()
+386 명령3()
+386 명령4()
486
+486 명령1()
+486 명령2()
+486 명령3()
+486 명령4()
586
+586 명령1()
+586 명령2()
+586 명령3()
+586 명령4()

Template Method

Template Method - 의도

- 전체 절차는 언제나 같다
- 몇몇 세부 작업은 그때 그때 다르다
- 예: Windows 어플리케이션의 파일 열기 절차

Template Method - 의도

- 예: Windows 어플리케이션의 파일 열기 절차:

- 메뉴 → File → Open → 대화상자 → 파일선택 →
  문서객체 생성 → 파일 읽기 → 화면에 문서 내용 표시하기

- 위 전체 절차는 언제나 같다

- 다음 세부 작업은 그때 그때 다르다
  - 문서 객체 생성
  - 파일 읽기
  - 화면에 문서 내용 표시하기

Template Method - 의도

- 공통적인 부분과 그때 그때 다른 부분을 잘 분리하여
  - 공통적인 부분의 재사용성을 향상시키고
  - 최소한의 코딩으로 그때 그때 다른 부분을 구현할 수 있으려면?

- 공통적인 전체 절차와 그때 그때 다른 단위 스텝

Template Method – 해결

- abstract class 를 만든다

- 전체 절차에 해당하는 메소드를 만든다
  - 공통적인 부분
  - 단위 스텝에 해당하는 메소드를 호출하는 형태로 구현된다
  - concrete method 로 구현할 수 있다

- 단위 스텝에 해당하는 메소드를 만든다
  - 세부 작업 내용이 공통적이라면 concrete method 로 구현
  - 세부 작업 내용이 그때 그때 달라져야 한다면
    abstract method 로 선언한다

Template Method – 해결

- 메뉴 → File → Open → 대화상자 → 파일선택 →
  문서객체 생성 → 파일 읽기 → 화면에 문서 내용 표시하기

- 공통적인 단위 스텝
  - 메뉴 → File → Open
  - 대화상자
  - 파일선택

- 매번 달라져야하는 단위 스텝
  - 문서객체 생성
  - 파일 읽기
  - 화면에 문서 내용 표시하기

Template Method – 해결

- 역할별로 분리하여 클래스를 만든다

- Application
- View
- Document

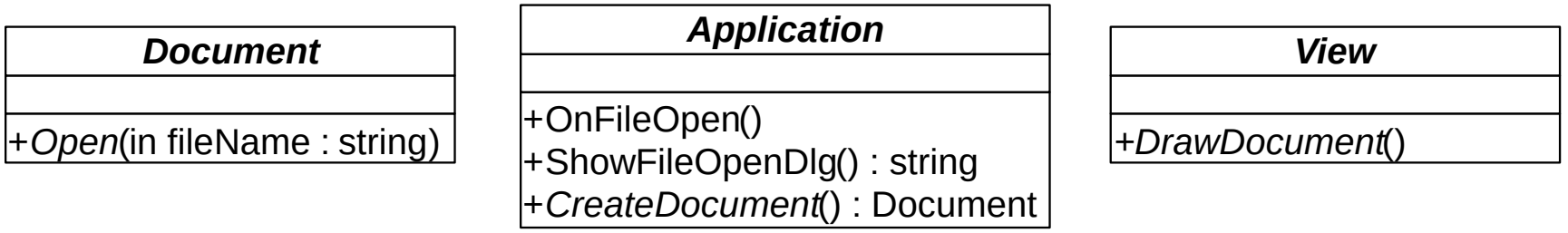
Template Method – 해결

- Application 클래스의 역할
  - 메뉴 → File → Open
  - 대화상자
  - 파일선택
  - 문서 객체 생성

- Document 클래스의 역할
  - 파일 읽기

- View 클래스의 역할
  - 화면에 문서 내용 표시하기

Template Method – 구조

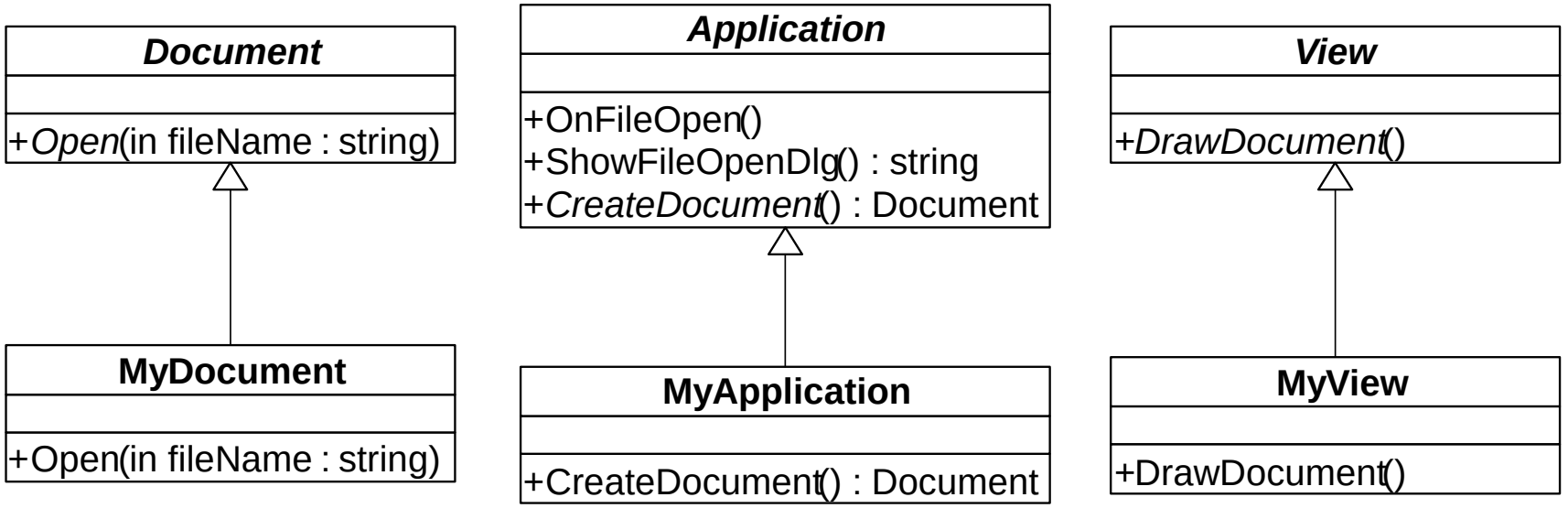



```
class Application  {
   public void OnFileOpen() {
        string fname = ShowFileOpenDlg();
        Document doc = CreateDocument();
        doc.Open(fname);
        view.DrawDocument();
   }
```

Template Method – 구조

- 공통적인 전체 절차
  - → Application.OnFileNew() 메소드로 구현
- 메뉴, 툴바, 단축키를 눌렀을 때 OnFileNew() 메소드를 호출하는 기능도 구현
- 공통적인 단위스텝 Application.ShowFileOpenDlg() 구현
- 매번 달라지는 단위스텝 → abstract method 로 선언
  - Application.CreateDocument()
  - Document.Open()
  - Document.DrawDocument()

Template Method – 구조




```
class MyApplication {
    public Document CreateDocument () {
        return new MyDocument();
    }
```

Template Method - 구조

- Document, Application, View 는
  클래스 라이브러리의 재사용 클래스

- 어플리케이션을 개발할 때는 위 클래스드들을 상속받아
  MyDocument, MyApplication, MyView 를 정의하고

- abstract method 를 재정의한다

정렬(sorting) 문제

- 정렬하는 데이터가 무엇이든 quick sort 정렬 알고리즘은 동일하다
- 그래서 quick sort 알고리즘을 미리 구현하여 재사용할 수 있다
- 미리 구현할 수 없는 부분
  - 데이터의 종류마다 비교하는 방법이 다르다
- quick sort 알고리즘을 재사용한 형태로 구현하려면?

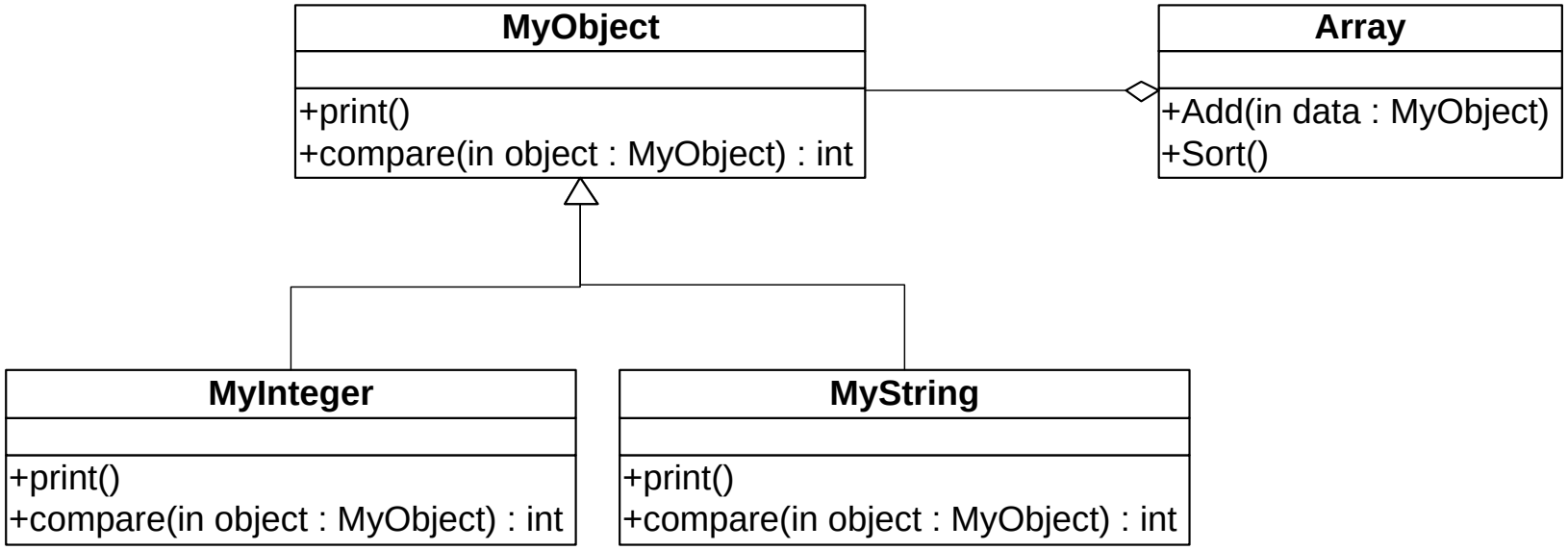
Template Method – 예

- 배열의 정렬

- 전체 절차는 언제나 같다
  - quick sort 알고리즘 구현

- 몇몇 세부 작업은 그때 그때 다르다
  - 크고 작음 비교: abstract method 선언

Template Method - 예



- 정렬 로직은 Sort() 메소드로 구현
- 비교하는 방법은 compare() 메소드로 구현
- Sort() 는 compare() 를 호출하여 원소를 비교한다

예제코드 분석

- template.e1
  - quick sort C 구현
  - 미리 구현할 수 없는 부분을 callback으로 구현

- template.e2
  - quick sort Java 구현
  - 미리 구현할 수 없는 부분을 abstract method로 구현

Template Method 패턴 요약

- 단위 스텝 메소드를 호출하는 형태로, 부모 클래스에 전체 절차 메소드를 구현한다
- 구현할 수 없는 단위 스텝 메소드는 abstract method로 선언한다
- 이 abstract method를 자식 클래스에서 구현한다.

예제 코드 분석

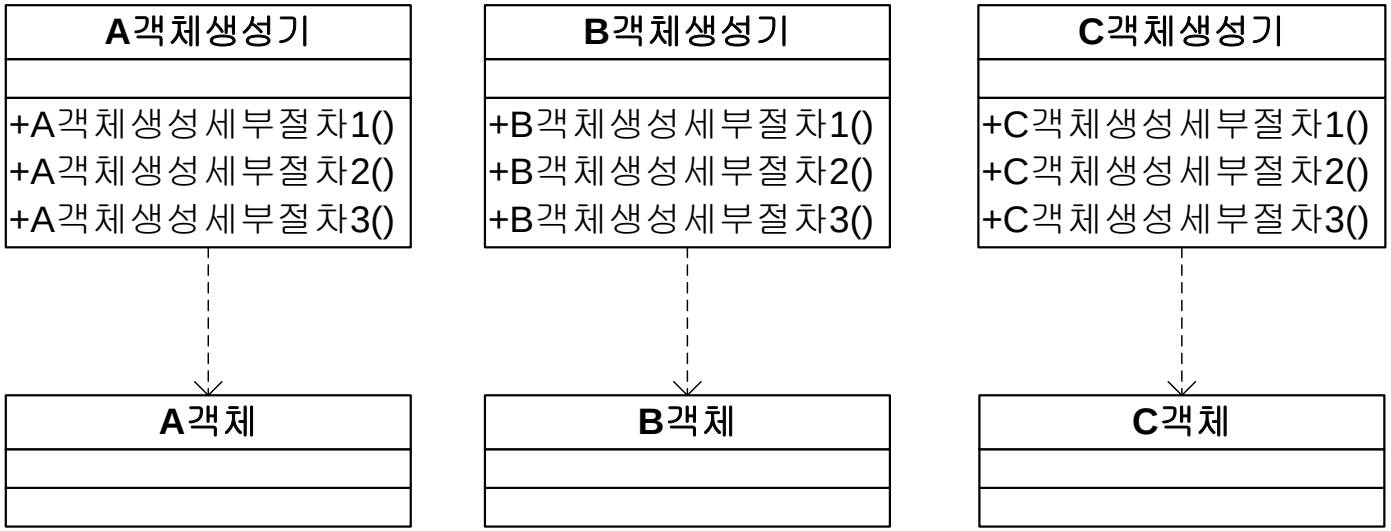
- decorator.e2 예제
  - 모든 도형 클래스에 그림자/핸들 장식 기능 추가하기
  - 도형 클래스를 수정하지 않고 기능 추가 → decorator 패턴

- template.e3 예제
  - 모든 도형 클래스에 그림자/핸들 장식 기능 추가하기
  - 도형 클래스를 수정해도 된다면? → template method 패턴

Builder

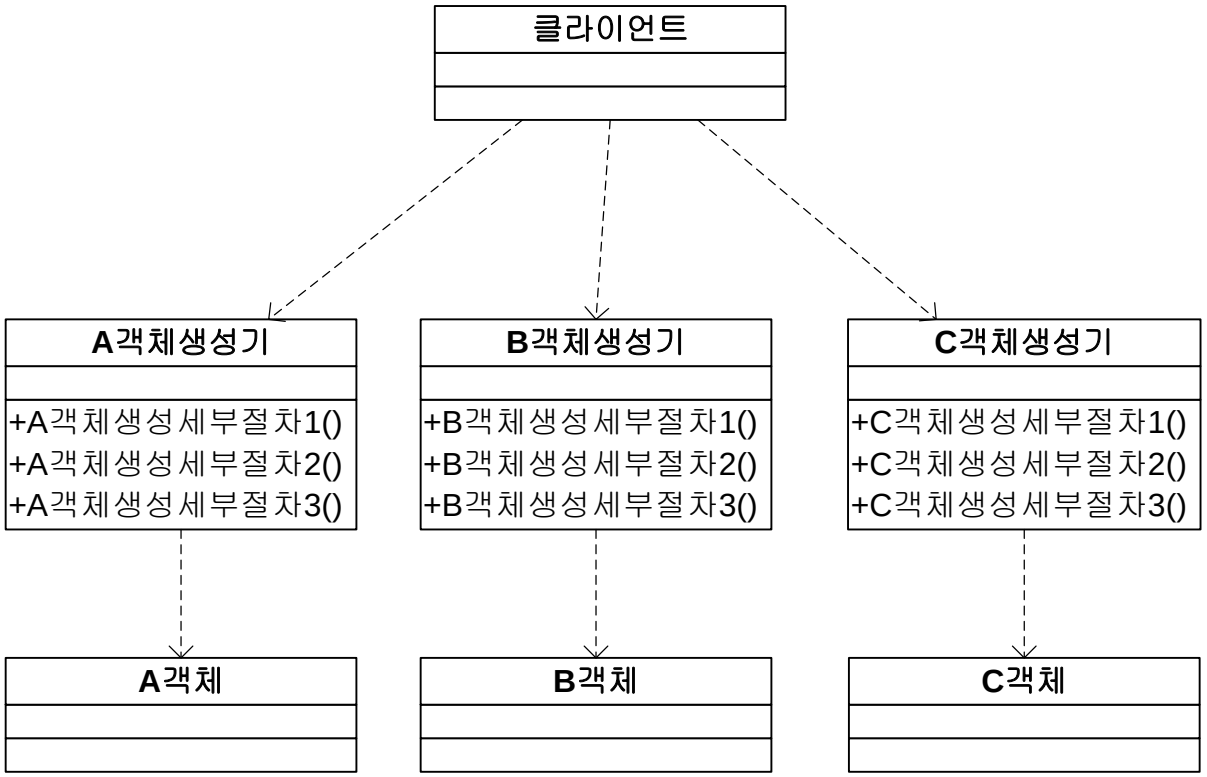
Builder – 문제

- 생성 절차가 긴 객체들이 있다
- 이들 객체의 생성 세부 절차는 상당히 유사하다



Builder – 문제

- 클라이언트에서 A, B, C객체를 생성하려면
  각각 A객체생성기, B객체생성기, C객체생성기를 사용하여
  객체를 생성하는 코드를 구현하여야 한다.



Builder – 문제

- 이들 객체의 생성 세부 절차는 상당히 유사하기 때문에
  클라이언트쪽의 A,B,C 객체 생성 코드들도
  서로 상당히 유사할 것이다

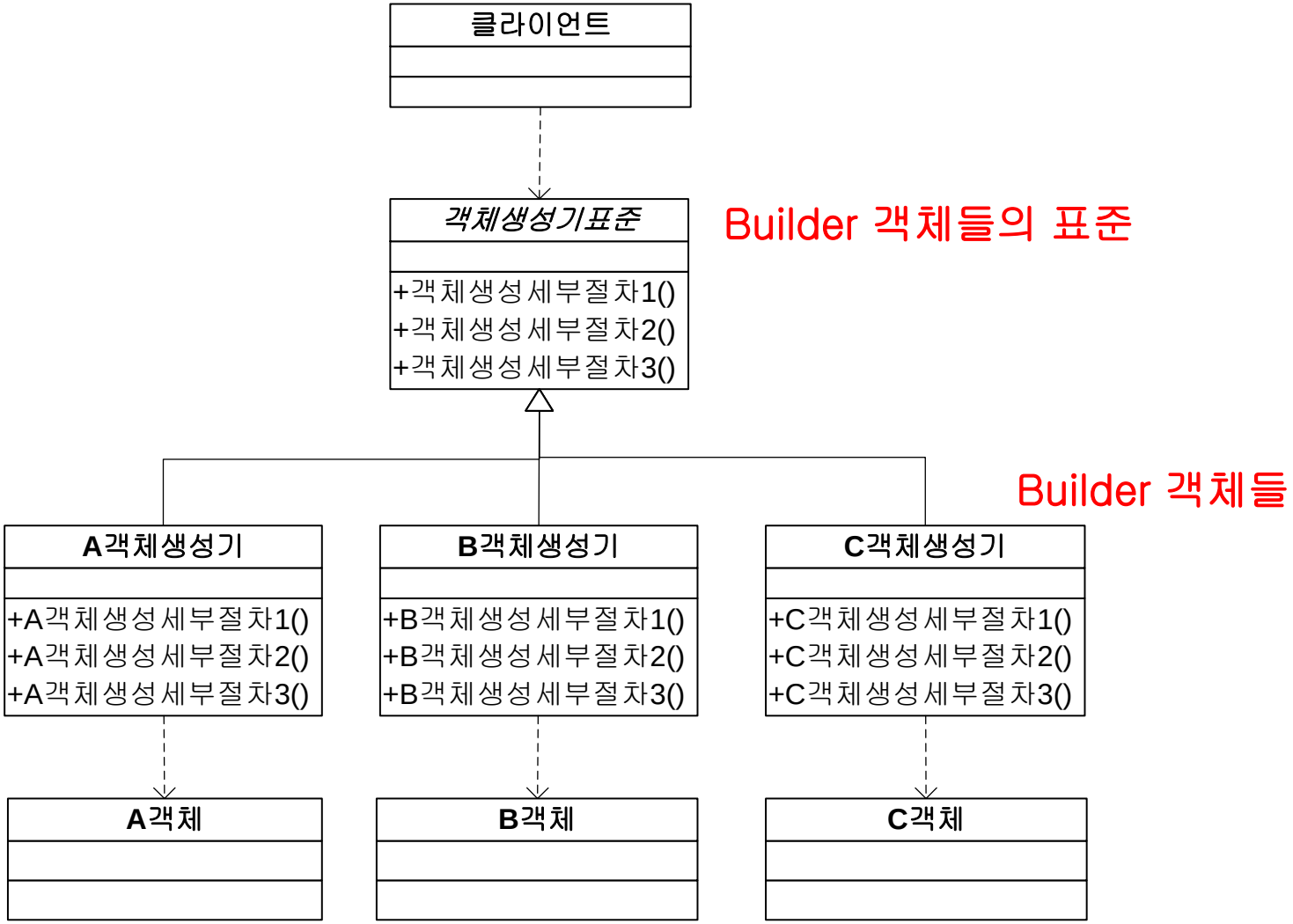
- 유사한 코드의 반복을 제거하는 방법은?

Builder – 해결

- 객체 생성기들의 표준을 만들면
  이 표준만 참조하여
  클라이언트에서 객체를 생성할 수 있다

- 클라이언트쪽 코드는
  구체적으로 어떤 객체를 만드는지와 무관하게 구현될 수 있다

Builder – 구조

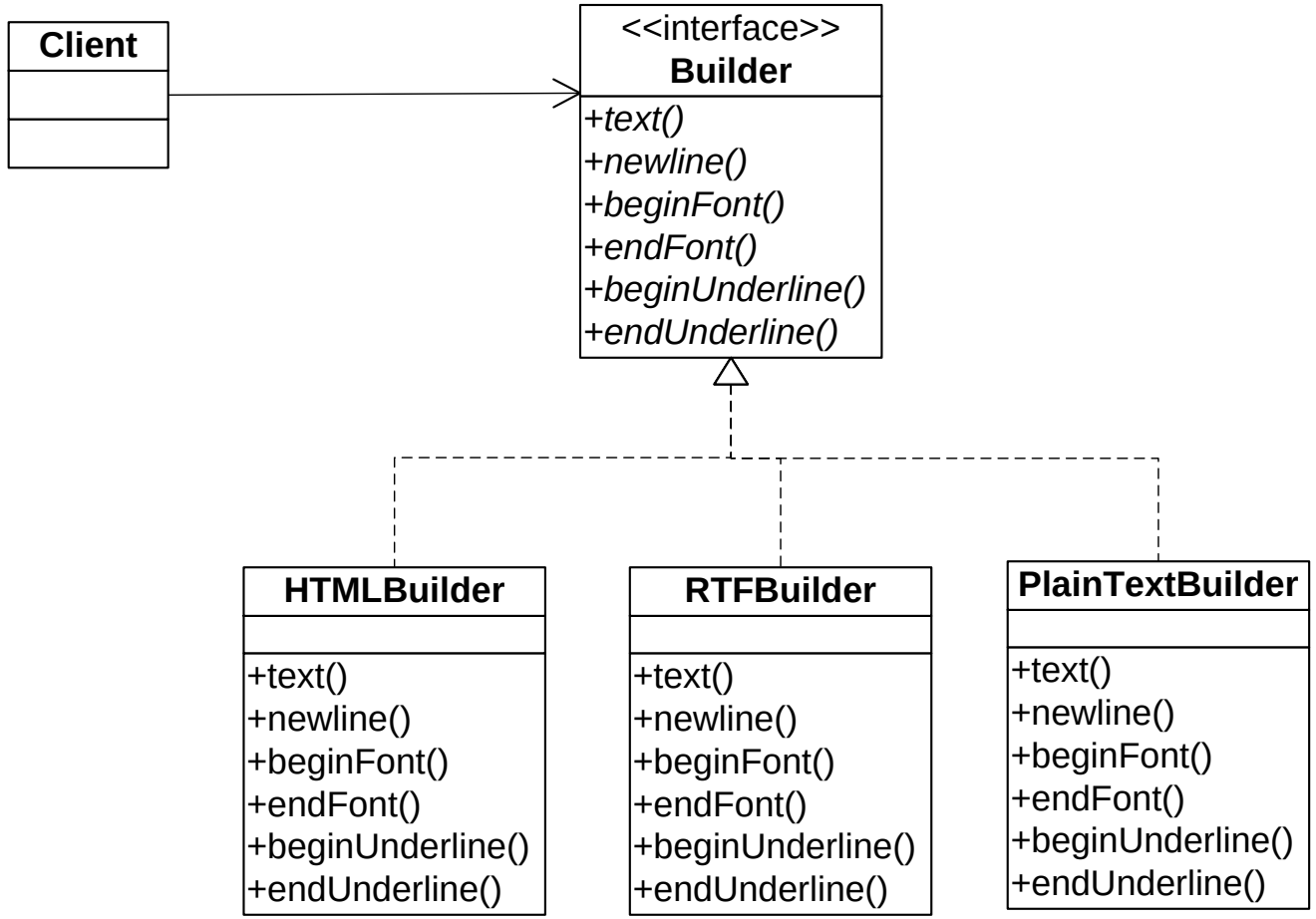

클라이언트
객체생성기표준
+객체생성세부절차1()
+객체생성세부절차2()
+객체생성세부절차3()
Builder 객체들의 표준
Builder 객체들
A객체생성기
+A객체생성세부절차1()
+A객체생성세부절차2()
+A객체생성세부절차3()
B객체생성기
+B객체생성세부절차1()
+B객체생성세부절차2()
+B객체생성세부절차3()
C객체생성기
+C객체생성세부절차1()
+C객체생성세부절차2()
+C객체생성세부절차3()
A객체
B객체
C객체

Builder – 결과

- 긴 생성 절차의 비슷한 객체들을 생성할 때
  클라이언트는 Builder 표준만 참조하여 구현하면 된다

- 클라이언트와 연결된 Builder 객체가 무엇이냐에 따라
  A, B, C 객체가 생성될 수 있다

Builder 패턴


Client
<<interface>>
Builder
+text()
+newline()
+beginFont()
+endFont()
+beginUnderline()
+endUnderline()
HTMLBuilder
+text()
+newline()
+beginFont()
+endFont()
+beginUnderline()
+endUnderline()
RTFBuilder
+text()
+newline()
+beginFont()
+endFont()
+beginUnderline()
+endUnderline()
PlainTextBuilder
+text()
+newline()
+beginFont()
+endFont()
+beginUnderline()
+endUnderline()

Builder – 적용

- Builder 패턴이 적용되려면

- 생성 절차가 긴 객체들
  - 절차적 로직에 의해서 객체의 내부구조를 정의함

- 이들 객체의 생성 세부 절차는 상당히 유사

예제 코드 분석

- builder.e3 → builder.e4
  - 빌더 패턴 적용

구현 실습:

- builder.e4 → builder.e4a
  - 주민등록번호 자동 치환 기능 구현함
  - 주민등록번호 뒷자리를 **** 문자로 치환하는 기능

- builder.e4a 코드 중복 문제
  - 주민등록번호 치환 기능을 구현한 코드가 모든 빌드 클래스에 반복된다
  - 새 builder 클래스를 추가 구현할 때 마다, 이 코드 중복이 반복된다
  - 전화번호 치환, 학번 치환 기능을 추가할 때,
    모든 builder 클래스를 수정해야 함.

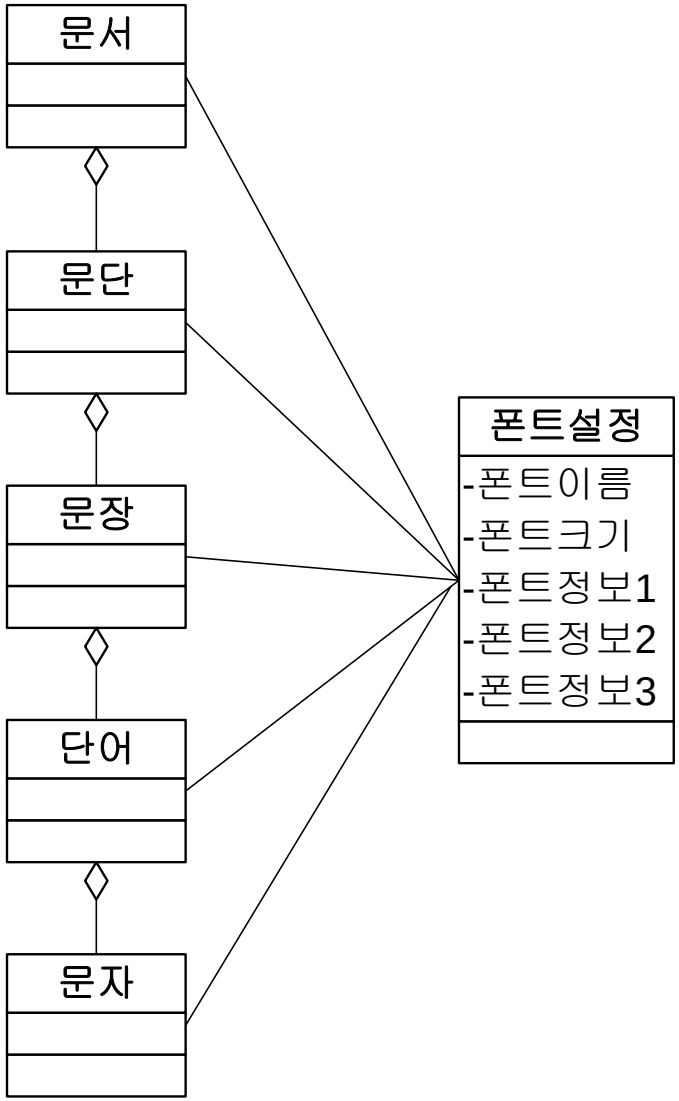
- 이 코드 중복을 제거하시오
  - 힌트: decorator 패턴 적용

Flyweight

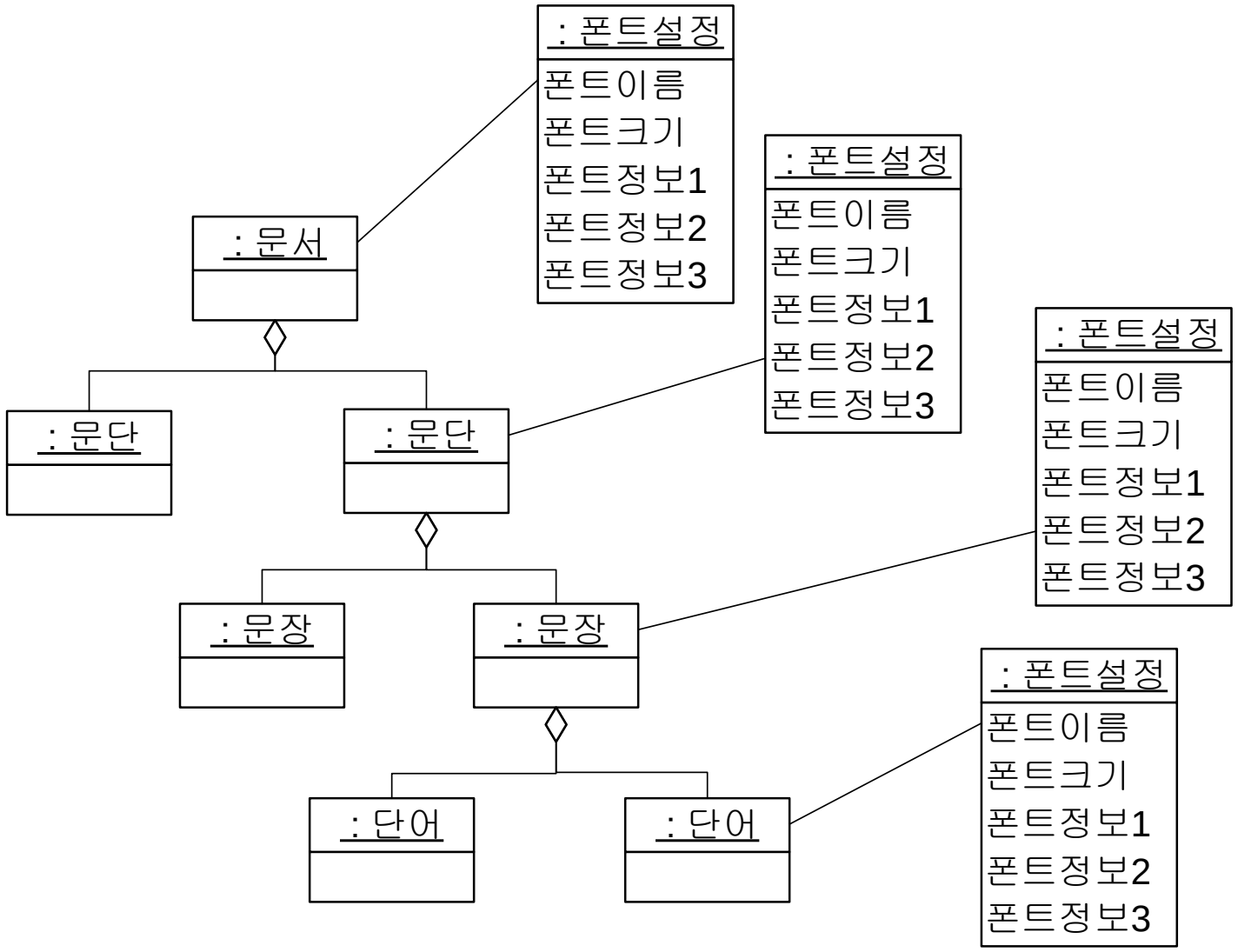
Flyweight – 문제

- 워드프로세서를 개발하고 있다
- 문단, 문장, 단어, 문자 → 모두 객체
- 문단, 문장, 단어, 문자 단위로 폰트 설정이 가능하다
- 폰트 설정
  - 폰트 이름
  - 폰트 크기
  - 그 폰트에 대한 정보
    - 각 문자의 넓이
    - 적절한 문자 간격
- 폰트 설정 정보의 중복

클래스 다이어그램



객체 다이어그램

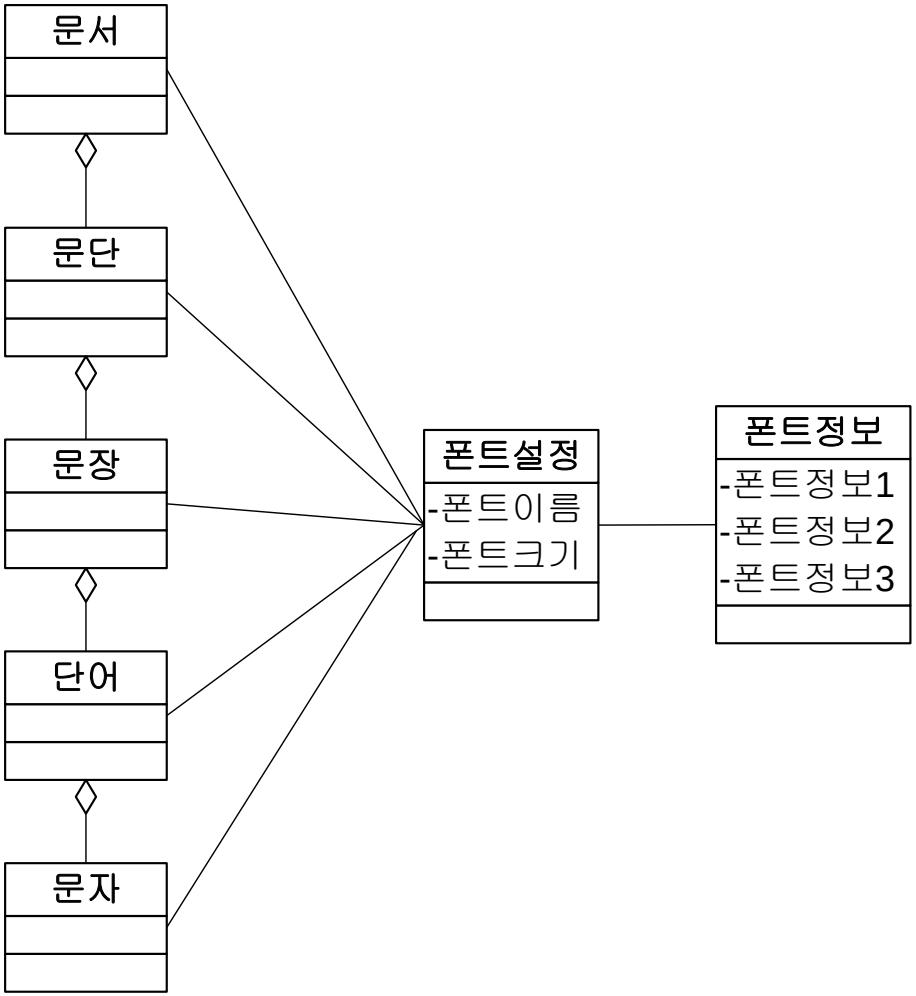

:폰트설정
폰트이름
폰트크기
폰트정보1
폰트정보2
폰트정보3
:문서
:폰트설정
폰트이름
폰트크기
폰트정보1
폰트정보2
폰트정보3
:문단
:문단
:폰트설정
폰트이름
폰트크기
폰트정보1
폰트정보2
폰트정보3
:문장
:문장
:폰트설정
폰트이름
폰트크기
폰트정보1
폰트정보2
폰트정보3
:단어
:단어

Flyweight – 해결

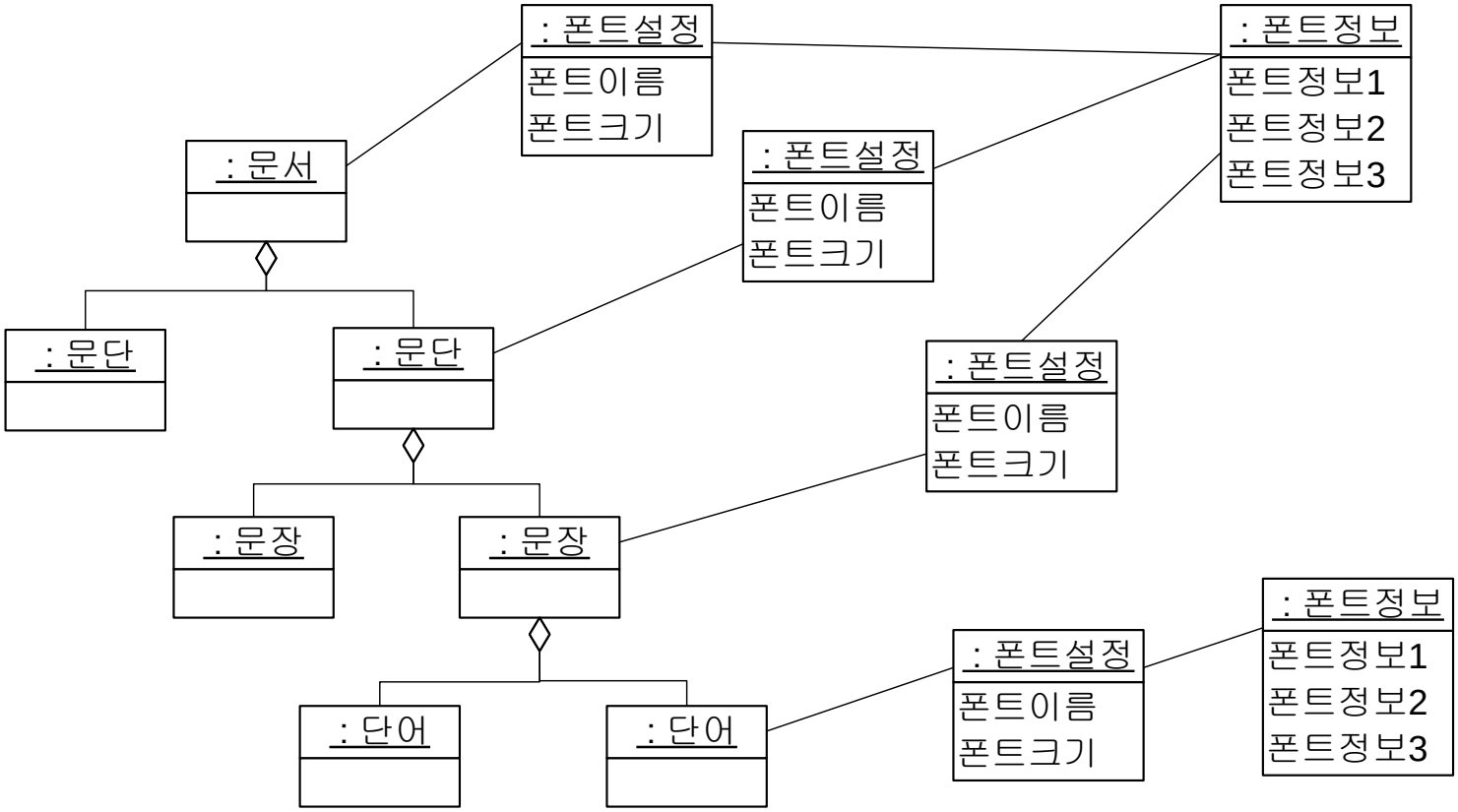
- 폰트 설정에서
  - 폰트 이름, 폰트 크기는 설정마다 다르지만
  - 폰트 정보는 공유 가능한 read only 값

- Flyweight 패턴
  - 객체 or 멤버 데이터를 중복해서 만들지 말고
  - 가능한 공유하자

클래스 다이어그램



객체 다이어그램


: 폰트설정
폰트이름
폰트크기
: 폰트정보
폰트정보1
폰트정보2
폰트정보3
: 문서
: 폰트설정
폰트이름
폰트크기
: 문단
: 문단
: 폰트설정
폰트이름
폰트크기
: 문장
: 문장
: 폰트정보
폰트정보1
폰트정보2
폰트정보3
: 폰트설정
폰트이름
폰트크기
: 단어
: 단어

Strategy

Strategy - 문제

- 자료구조에 집합연산을 구현하려고 한다
  - 교집합, 합집합, 차집합

- 교집합을 구현하려면
  동일한 원소를 찾는 알고리즘을 선택해야 한다

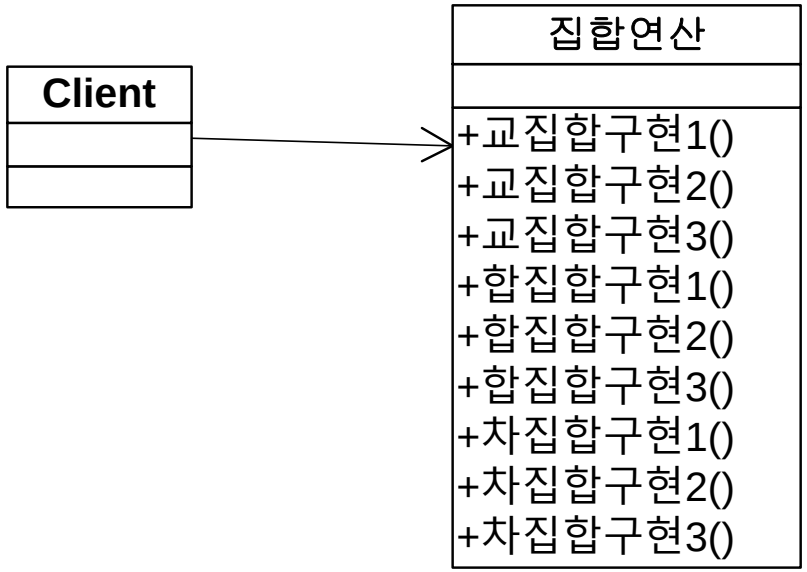
- 상황에 따라 적절한 알고리즘을 적용할 수 있도록
  여러개의 알고리즘을 구현하려고 한다
  - 원소가 적다면 선형탐색
  - 원소가 많다면 이진탐색
  - 먼저 정렬한 후 이진탐색
  - 이진트리를 이용한 이진탐색

Strategy - 문제

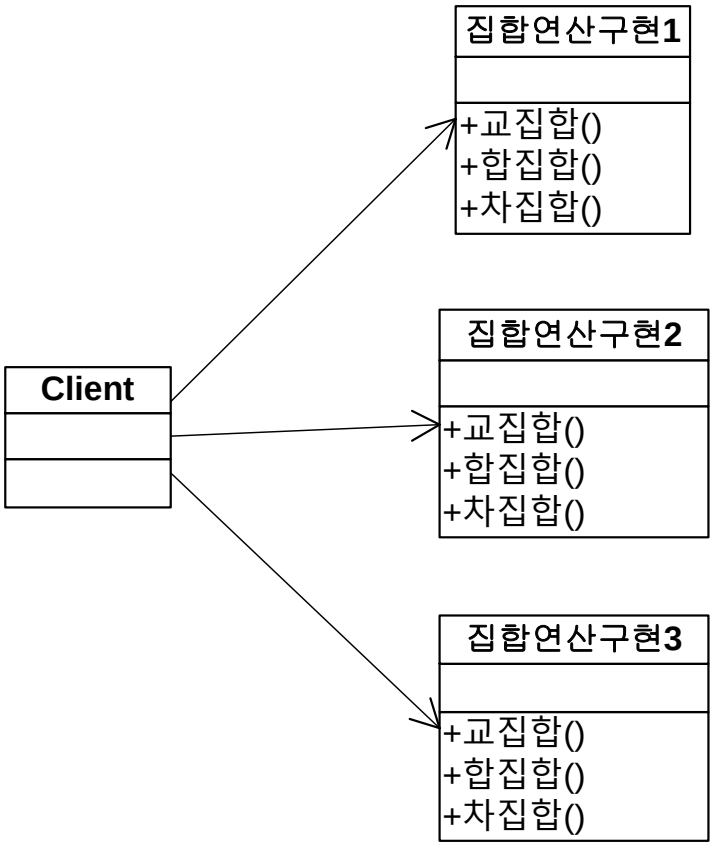
- 위 알고리즘들을 자료구조 클래스에 모두 구현하는 것은 어떨까?
  - 알고리즘 당 하나의 메소드로 구현
  - 집합 연산 x 알고리즘 = 메소드 수
  - 메소드가 너무 많다
  - 많은 메소드를 사용해야 하는 클라이언트 측 코드도 복잡해짐

- 어떻게 분리하는 것이 바람직할까?

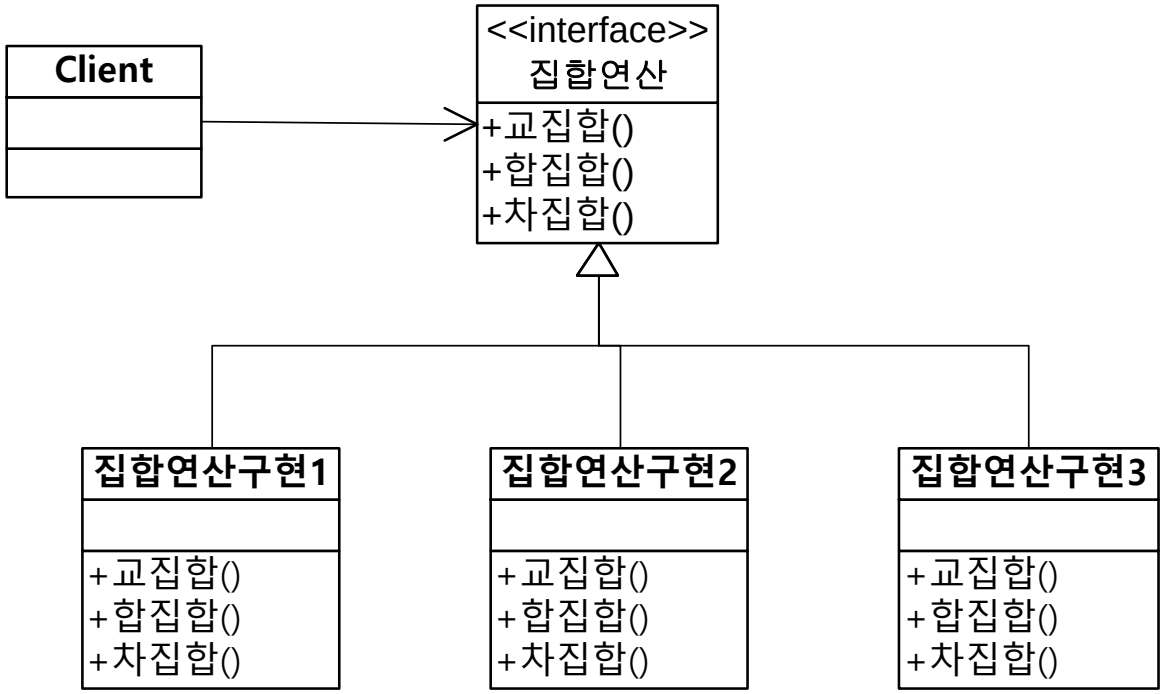
Strategy - 문제


Client
집합연산
+교집합구현1()
+교집합구현2()
+교집합구현3()
+합집합구현1()
+합집합구현2()
+합집합구현3()
+차집합구현1()
+차집합구현2()
+차집합구현3()

클래스 분리


집합연산구현1
+교집합()
+합집합()
+차집합()
집합연산구현2
+교집합()
+합집합()
+차집합()
Client
집합연산구현3
+교집합()
+합집합()
+차집합()

다형성 구현


Client
<<interface>>
집합연산
+교집합()
+합집합()
+차집합()
집합연산구현1
+교집합()
+합집합()
+차집합()
집합연산구현2
+교집합()
+합집합()
+차집합()
집합연산구현3
+교집합()
+합집합()
+차집합()

적당한 구현 선택

- 앞 슬라이드 형태로 구현했다면,
  - 상황에 따라 적당한 집합연산구현 클래스를 선택하고
  - 그 클래스의 객체를 생성하여 사용하면 된다.

실행 도중 전략 변경

- 실행도중 상황이 변해서 집합연산구현 클래스를 변경해야 한다면?

- 클래스 변경
  - 적당한 새 집합연산구현 클래스 객체를 생성하고
  - 구 집합 객체에 들어있던 자료를 새 집합 객체로 옮기고
  - 새 집합 객체를 사용한다

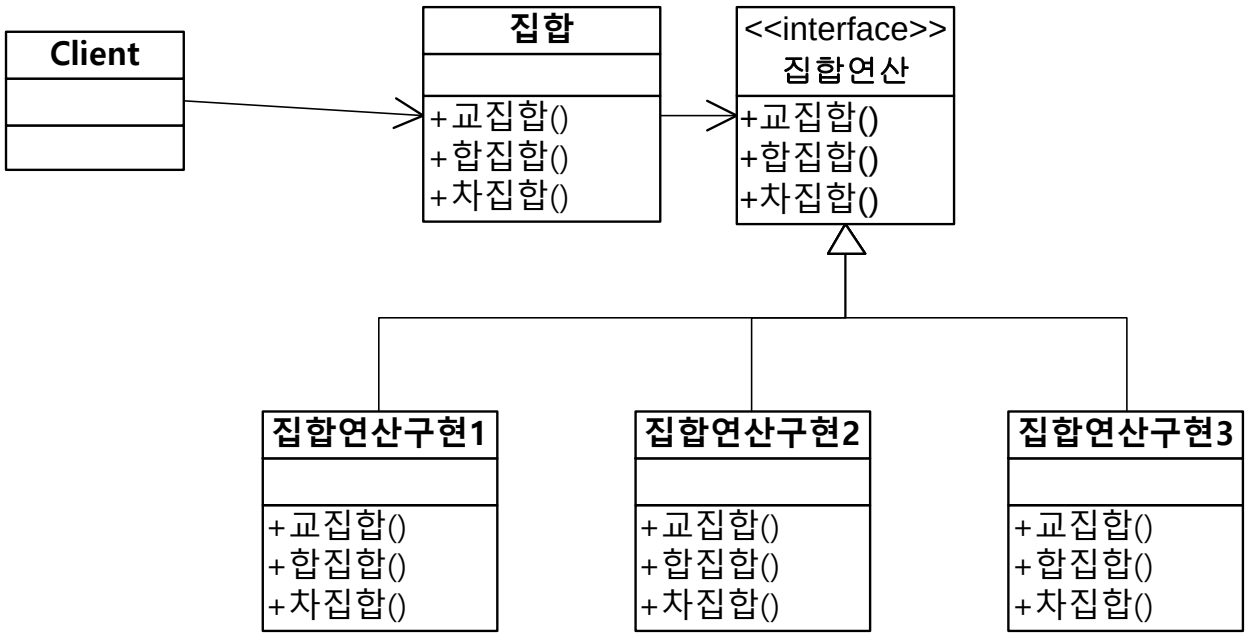
- 이 클래스 변경 작업을 어디에 구현해야 하나?
  - 클라이언트?
  - 집합연산구현 클래스?

실행 도중 전략 변경

- 실행 도중 전략 변경 코드를 wrapper 클래스로 추출

- 이 wrapper 클래스는
  클라이언트와 집합연산구현 클래스 사이에서 전략 변경만 수행하고
  나머지 작업은 집합연산구현 클래스에 위임

Strategy – 구조


Client
집합
+교집합()
+합집합()
+차집합()
<<interface>>
집합연산
+교집합()
+합집합()
+차집합()
집합연산구현1
+교집합()
+합집합()
+차집합()
집합연산구현2
+교집합()
+합집합()
+차집합()
집합연산구현3
+교집합()
+합집합()
+차집합()

Strategy – 구조

- 적절한 집합연산구현 클래스의 객체를 하나 생성하여
  집합 클래스 객체에 연결한다

- 집합 클래스의 집합연산 메소드는 연결된 집합연산구현 클래스를 호출하는 형태로 구현됨 (위임)

- 집합 클래스 객체에 연결된 집합연산구현 객체를 교체하면 집합연산 구현 방식이 변경됨

Strategy – 결과

- 클라이언트는 많은 집합연산구현에 무관하고, 집합 클래스만 참조하여 구현됨
- 객체들을 생성할 때 적절한 집합연산구현 객체를 생성하여 집합 클래스 객체에 연결하면 됨.
- 집합연산구현 클래스가 추가될 때 클라이언트 코드는 영향을 받지 않는다
- 실행시 집합연산구현 객체가 다른 객체로 변경되어도 클라이언트는 영향을 받지 않는다

예제 코드 분석

- strategy.e1

- Strategy 패턴과 유사한 구조의 패턴은 무엇인가? 공통점과 차이점은?

Memento

Memento - 의도

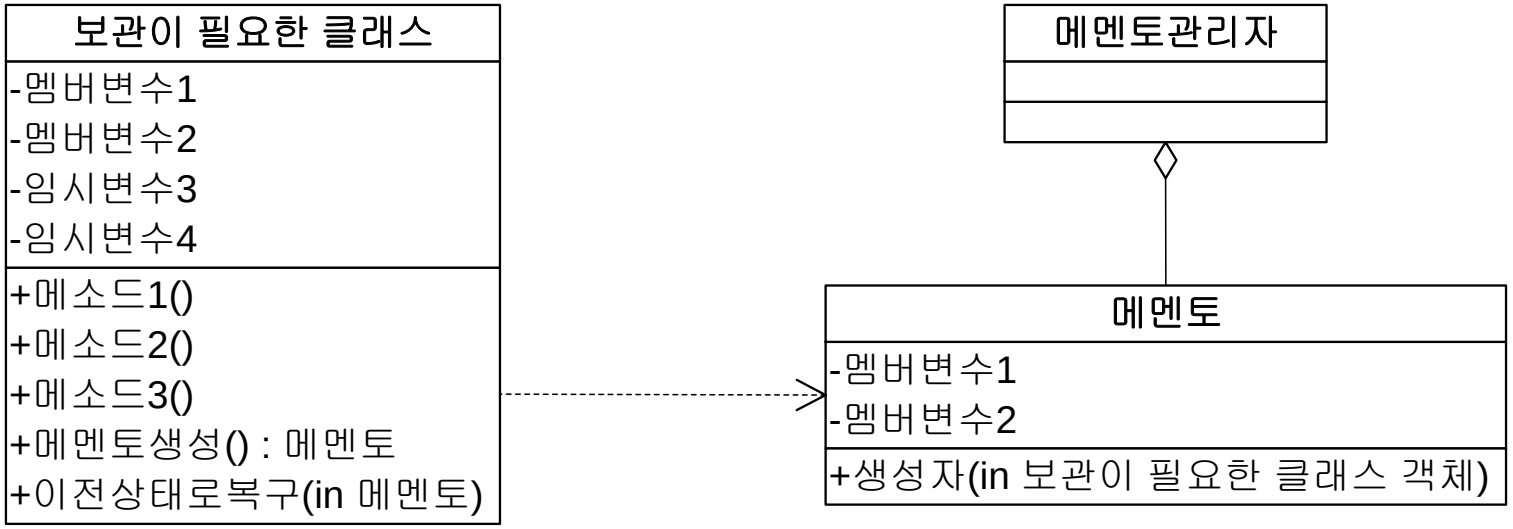
- 객체를 복사해서 보관하는 것이 필요한 경우가 있다
  - 데이터베이스의 snapshot
  - undo 구현

- 계산 과정에서만 잠시 필요한 멤버 변수는 복사할 필요가 없다

- 이런 임시 멤버 변수가 많다면 객체를 복사하는 것은 메모리 낭비

- 보관해야하는 멤버 변수들만 포함하는 새 클래스를 정의하자
  - 메멘토 객체

Memento – 구조


보관이 필요한 클래스
-멤버변수1
-멤버변수2
-임시변수3
-임시변수4
+메소드1()
+메소드2()
+메소드3()
+메멘토생성() : 메멘토
+이전상태로복구(in 메멘토)
메멘토관리자
메멘토
-멤버변수1
-멤버변수2
+생성자(in 보관이 필요한 클래스 객체)

Binary Tree Iterator 구현 실습

- 트리 구조에대한 선형 탐색은 stack 자료구조를 사용하여 구현해야 함.

- ArrayDeque, LinkedList 클래스를 stack 처럼 사용할 수 있다.
  - push(value), pop() 메소드

- example1 의 binary tree 에 대한 iterator를 구현하시오