14. 컴퍼지트 패턴

학습목표

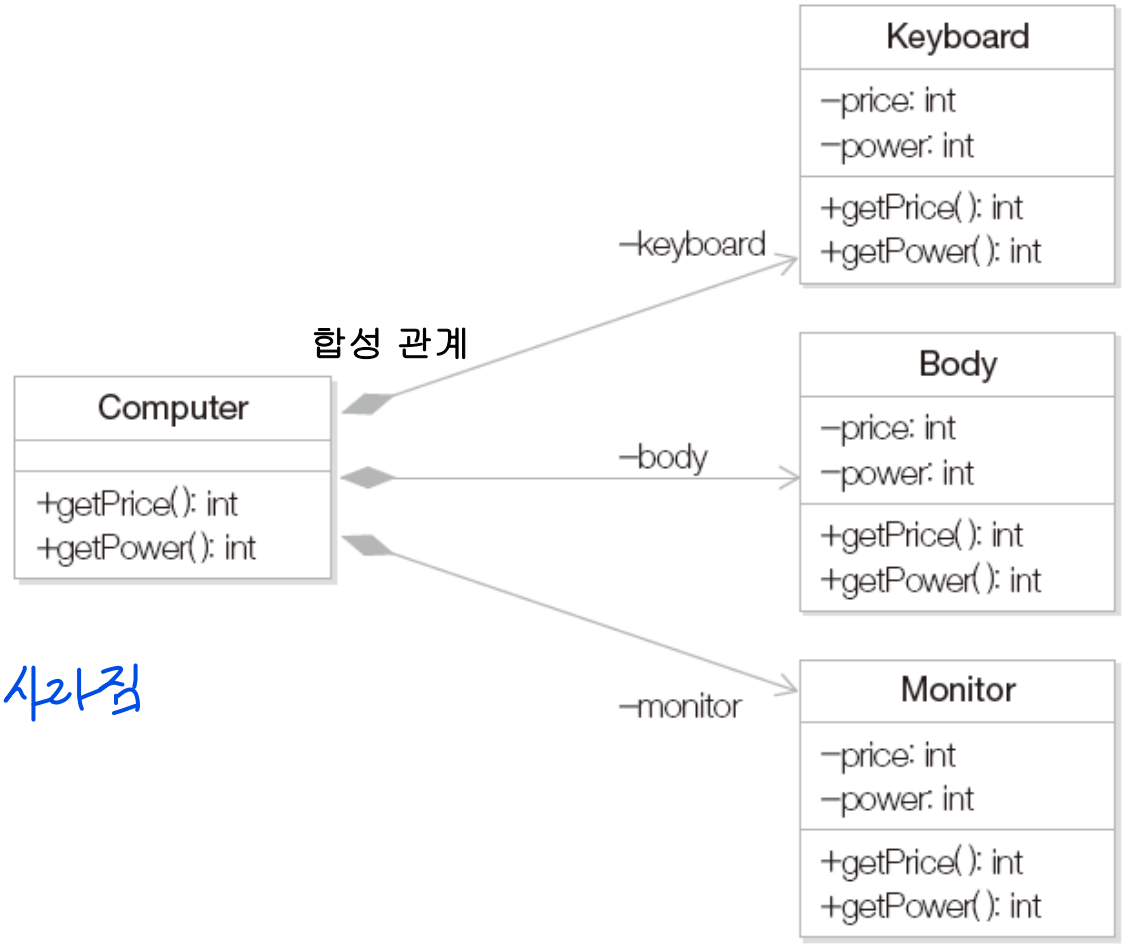
학습목표

- 부분–전체의 관계가 있는 객체의 설계 방법 이해하기
- 컴퍼지트 패턴을 이용한 부분–전체 객체의 설계 방법 이해하기
- 사례 연구를 통한 컴퍼지트 패턴의 핵심 특징 이해하기

14.1 컴퓨터에 추가 장치 지원하기

그림 14-1 컴퓨터의 구성

그림 14-2 Computer 클래스의 클래스 다이어그램



Computer 사라지면 나머지도 사라짐

소스 코드

코드 14-1

```
public class Keyboard {
   private int price ;
   private int power;
   public Keyboard(int power, int price) {
      this.power = power ; this.price = price ;
   }
   public int getPrice() { return price ; }
   public int getPower() { return power; }
}
```

```
public class Body {
   private int price ;
   private int power;
   public Body(int power, int price) {
      this.power = power ; this.price = price ;
   }
   public int getPrice() { return price ; }
   public int getPower() { return power; }
}
```

```
public class Monitor {
   private int price ;
   private int power;
   public Monitor(int power, int price) {
      this.power = power ; this.price = price ;
   }
   public int getPrice() { return price ; }
   public int getPower() { return power; }
}
```

소스 코드

코드 14-2

```
public class Computer {
   private Body body ;
   private Keyboard keyboard ;
   private Monitor monitor ;

   public void addBody(Body body) { this.body = body ; }
   public void addKeyboard(Keyboard keyboard) {
      this.keyboard = keyboard ;
   }
   public void addMonitor(Monitor monitor) { this.monitor = monitor ; }
   public int getPrice() {
      int bodyPrice = body.getPrice() ;
      int keyboardPrice = keyboard.getPrice() ;
      int monitorPrice = monitor.getPrice() ;
      return bodyPrice + keyboardPrice + monitorPrice ;
   }
   public int getPower() {
      int bodyPower = body.getPower() ;
      int keyboardPower = keyboard.getPower() ;
      int monitorPower = monitor.getPower() ;
      return bodyPower + keyboardPower + monitorPower ;
   }
}
```

클라이언트 소스 코드

코드 14-3

```
public class Client {
    public static void main(String[] args) {
        // 컴퓨터의 부품으로서 Body, Keyboard, Monitor 객체를 생성함
        Body body = new Body(100, 70) ; //전력소비량, 가격 순으로 인자 넘김
        Keyboard keyboard = new Keyboard(5, 2) ;
        Monitor monitor = new Monitor(20, 30) ;

        // Computer 객체를 생성하고 부품 객체들을 설정함
        Computer computer = new Computer() ;
        computer.addBody(body) ;
        computer.addKeyboard(keyboard) ;
        computer.addMonitor(monitor) ;

        // 컴퓨터의 가격과 전력소비량을 구함
        int computerPrice = computer.getPrice() ;
        int computerPower = computer.getPower() ;
        System.out.println("Computer Power: " + computerPower + " W") ;
        System.out.println("Computer Price: " + computerPrice + " 만원") ;
    }
}
```

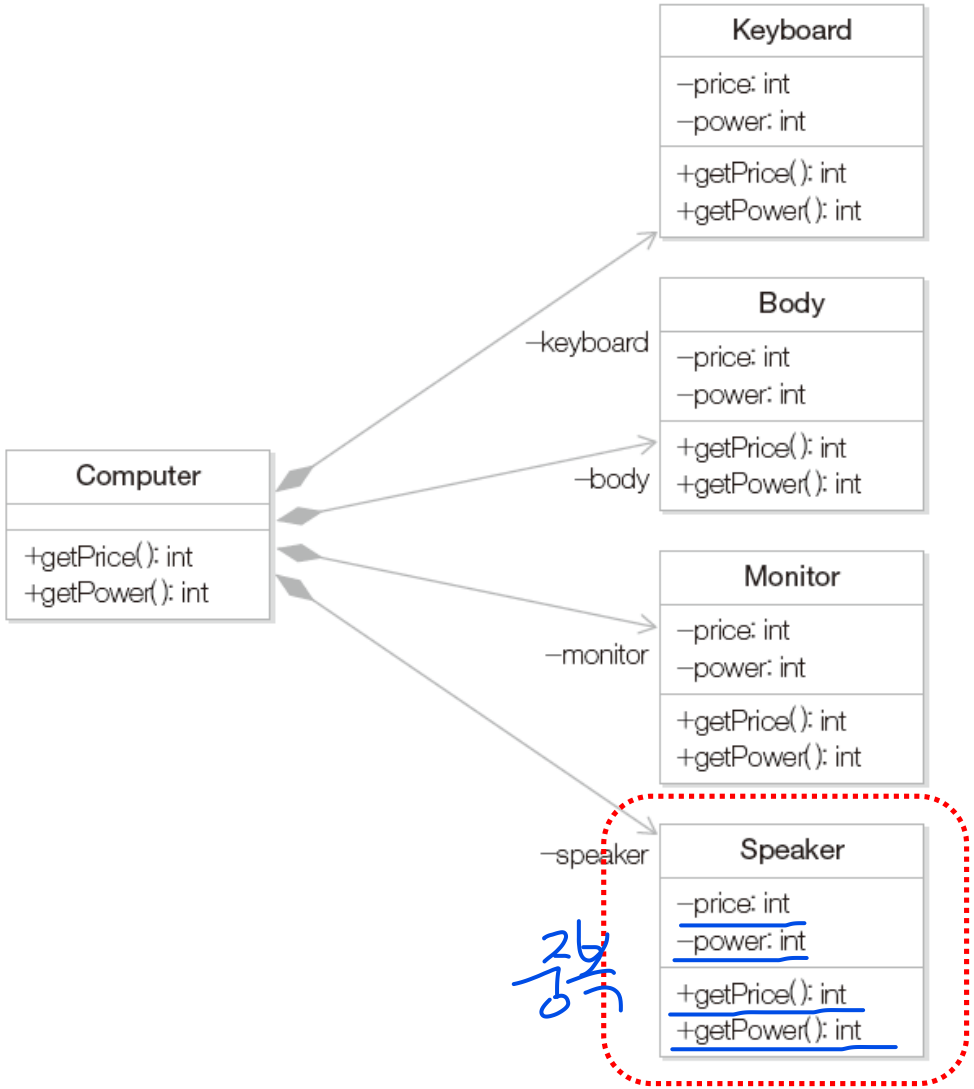
Computer Power: 125 W
Computer Price: 102 만원

14.2 문제점

- 만약 부품으로서 Speaker를 추가해야 한다면? 또는 Mouse를 추가한다면?

그림 14-3 스피커를 컴퓨터 부품으로 추가한 경우

그림 14-4 Speaker 클래스의 추가



스피커의 추가

코드 14-5

```
public class Speaker {
   private int price ;
   private int power;
   public Speaker(int power, int price) {
      this.power = power ;
      this.price = price ;
   }
   public int getPrice() { return price ; } //스피커의 가격
   public int getPower() { return power; }//스피커의 전력량
}
```

스피커의 추가

코드 14-6

스피커 객체를 가질 수 있도록 수정이 필요

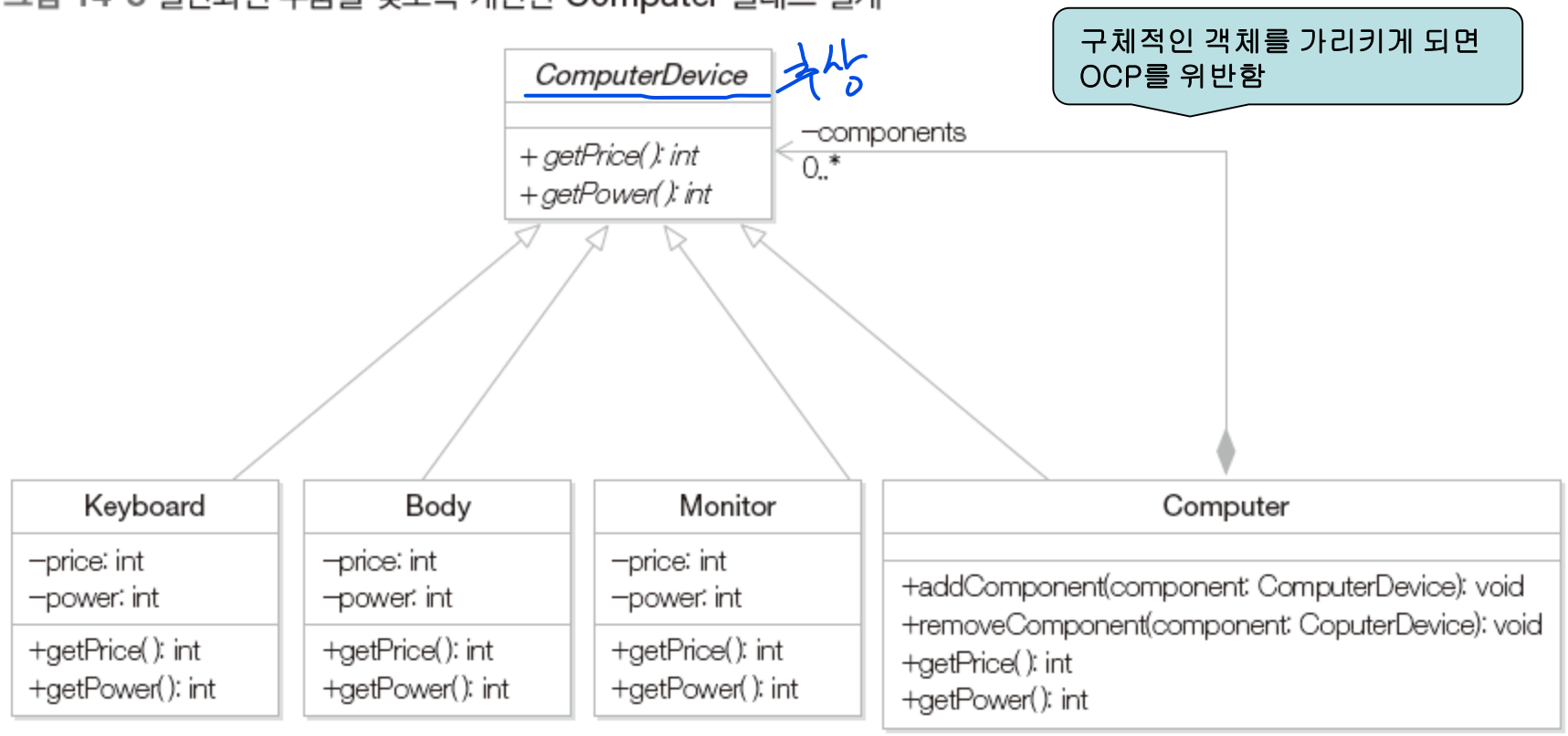
부품을 추가할 때마다 Computer 소스 코드를 수정해야 함 ➔ OCP를 위반함

```
public class Computer {
    private Body body ;
    private Keyboard keyboard ;
    private Monitor monitor ;
    private Speaker speaker ;
    public void addBody(Body body) { this.body = body ; }
    public void addKeyboard(Keyboard keyboard) { this.keyboard = keyboard ; }
    public void addMonitor(Monitor monitor) { this.monitor = monitor ; }
    public void addSpeaker(Speaker speaker) { this.speaker = speaker ; }
    public int getPrice() {
        int bodyPrice = body.getPrice() ;
        int keyboardPrice = keyboard.getPrice() ;
        int monitorPrice = monitor.getPrice() ;
        int speakerPrice = speaker.getPrice() ;
        return bodyPrice + keyboardPrice + monitorPrice  + speakerPrice ;
    }
    public int getPower() {
        int bodyPower = body.getPower() ;
        int keyboardPower = keyboard.getPower() ;
        int monitorPower = monitor.getPower() ;
        int speakerPower = speaker.getPower() ;
        return bodyPower + keyboardPower + monitorPower + speakerPower ;
    }
}
```

14.3. 해결책

❖ 구체적인 부품들을 일반화한 클래스를 정의하고 이를 가리키도록 설계

그림 14-5 일반화된 부품을 갖도록 개선한 Computer 클래스 설계



14.3. 해결책: 소스 코드

코드 14-6

```
public abstract class ComputerDevice {
   public abstract int getPrice() ;
   public abstract int getPower() ;
}
```

```
public class Keyboard extends ComputerDevice {
   private int price ;
   private int power;

   public Keyboard(int power, int price) {
      this.power = power ;
      this.price = price ;
   }
   public int getPrice() { return price ; }
   public int getPower() { return power; }
}
```

Body, Monitor 클래스도 비슷함

14.3. 해결책: 소스 코드

코드 14-7

Computer는 ComputerDevice의 서브클래스
동시에 다른 ComputerDevice를 가질 수 있음

```
public class Computer extends ComputerDevice {
   // 복수 개의 ComputerDevice를 가리킴
   private List<ComputerDevice> components = new ArrayList<ComputerDevice>() ;
   // ComputerDevice를 Computer에 추가
   public void addComponent(ComputerDevice component) {
      components.add(component) ;
   }
   // ComputerDevice를 Computer에서 제거
   public void removeComponent(ComputerDevice component) {
      components.remove(component) ;
   }
   public int getPrice() {
      int price = 0 ;
      for ( ComputerDevice component: components )
               price += component.getPrice() ;
      return price ;
   }
   public int getPower() {
      int power = 0 ;
      for ( ComputerDevice component: components )
               power += component.getPower() ;
      return power ;
   }
}
```

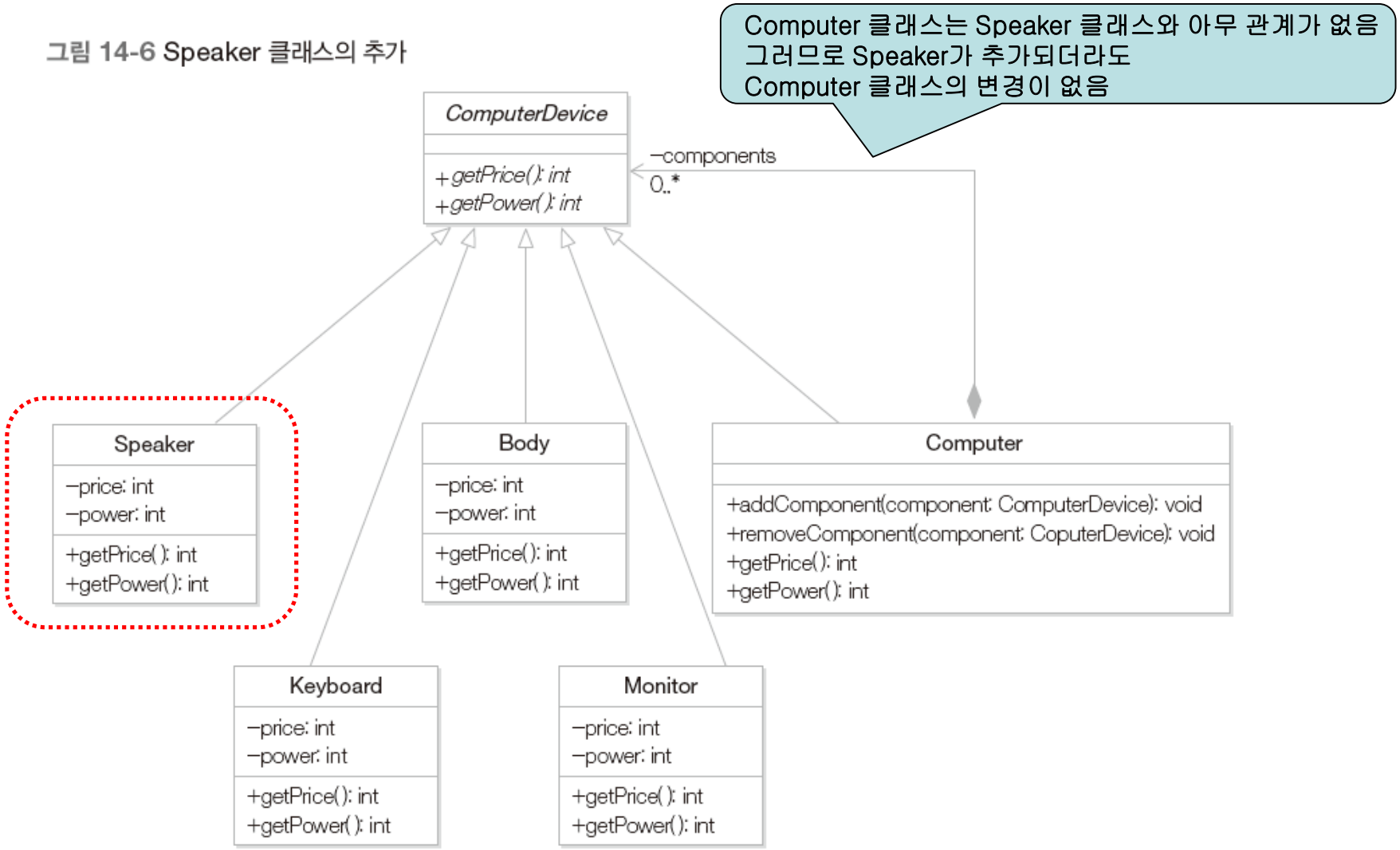
14.3. 해결책: 소스 코드

코드 14-8

```
public class Client {
   public static void main(String[] args) {
      // 컴퓨터의 부품으로서 Body, Keyboard, Monitor 객체를 생성함
      Body body = new Body(100, 70) ;
      Keyboard keyboard = new Keyboard(5, 2) ;
      Monitor monitor = new Monitor(20, 30) ;

      // Computer 객체를 생성하고 부품 객체들을 설정함
      Computer computer = new Computer() ;
      computer.addComponent(body) ;
      computer.addComponent(keyboard) ;
      computer.addComponent(monitor) ;

      int computerPrice = computer.getPrice() ;
      int computerPower = computer.getPower() ;
      System.out.println("Computer Power: " + computerPower + " W") ;
      System.out.println("Computer Price: " + computerPrice + " 만원") ;

   }
}
```

스피커의 추가

그림 14-6 Speaker 클래스의 추가



소스 코드

코드 14-9

```
public class Client {
   public static void main(String[] args) {
      // 컴퓨터의 부품으로서 Body, Keyboard, Monitor, Speaker 객체를 생성함
      Body body = new Body(100, 70) ;
      Keyboard keyboard = new Keyboard(5, 2) ;
      Monitor monitor = new Monitor(20, 30) ;
      Speaker speaker = new Speaker(10, 10) ;

      // Computer 객체를 생성하고 부품 객체들을 설정함
      Computer computer = new Computer() ;
      computer.addComponent(body) ;
      computer.addComponent(keyboard) ;
      computer.addComponent(monitor) ;
      computer.addComponent(speaker) ;

      int computerPrice = computer.getPrice() ;
      int computerPower = computer.getPower() ;
      System.out.println("Computer Power: " + computerPower + " W") ;
      System.out.println("Computer Price: " + computerPrice + " 만원") ;
   }
}
```

Computer Power: 135 W
Computer Price: 112 만원

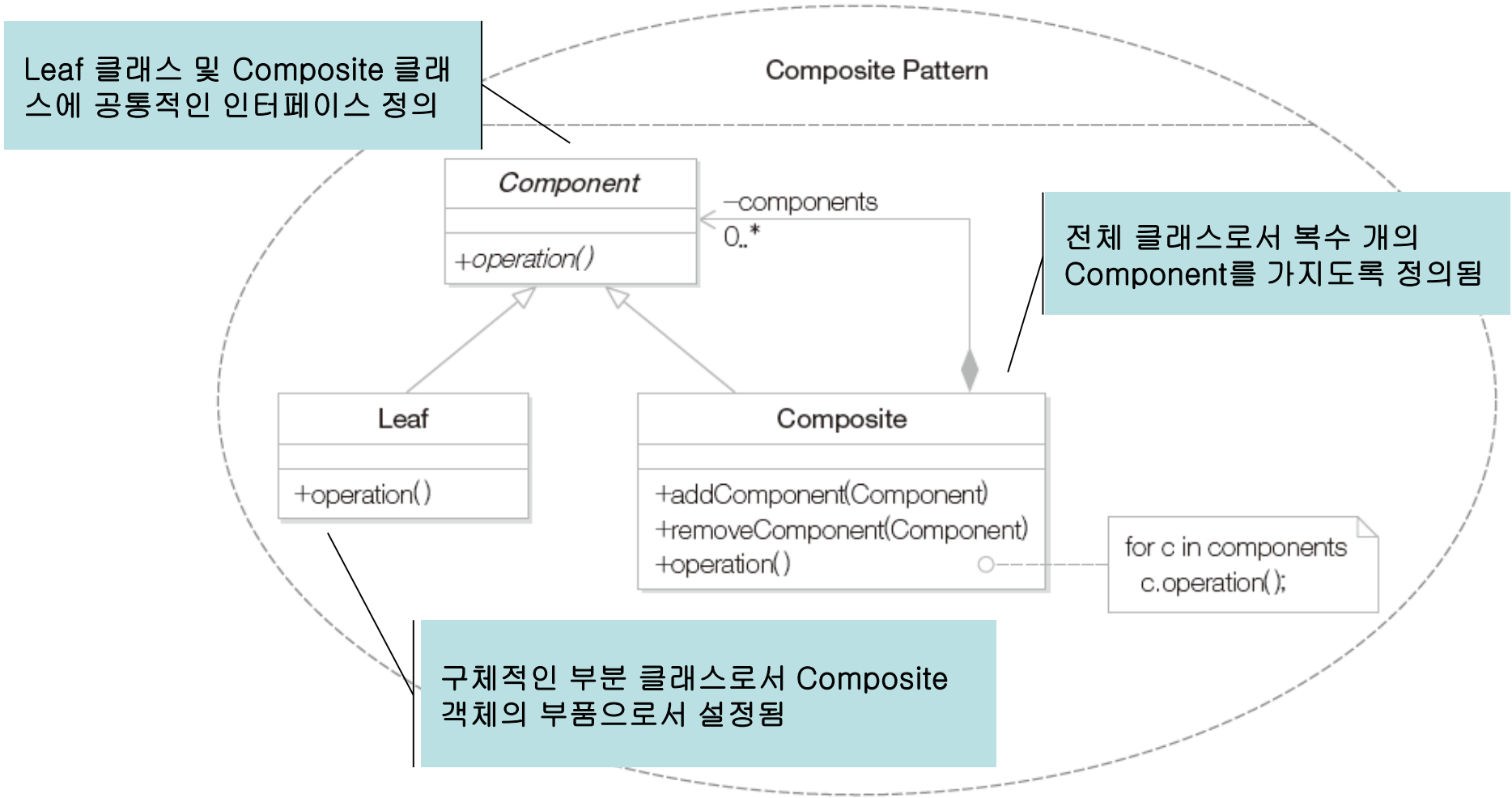
14.4 컴퍼지트 패턴

- 부분(part)-전체(whole)의 관계를 가지는 객체들을 정의할 때 유용

컴퍼지트 패턴은 전체-부분의 관계를 가지는 객체들 간의 관계를 정의할 때 유용하다. 그리고 클라이언트는 전체와 부분을 구분하지 않고 동일한 인터페이스를 사용할 수가 있다.

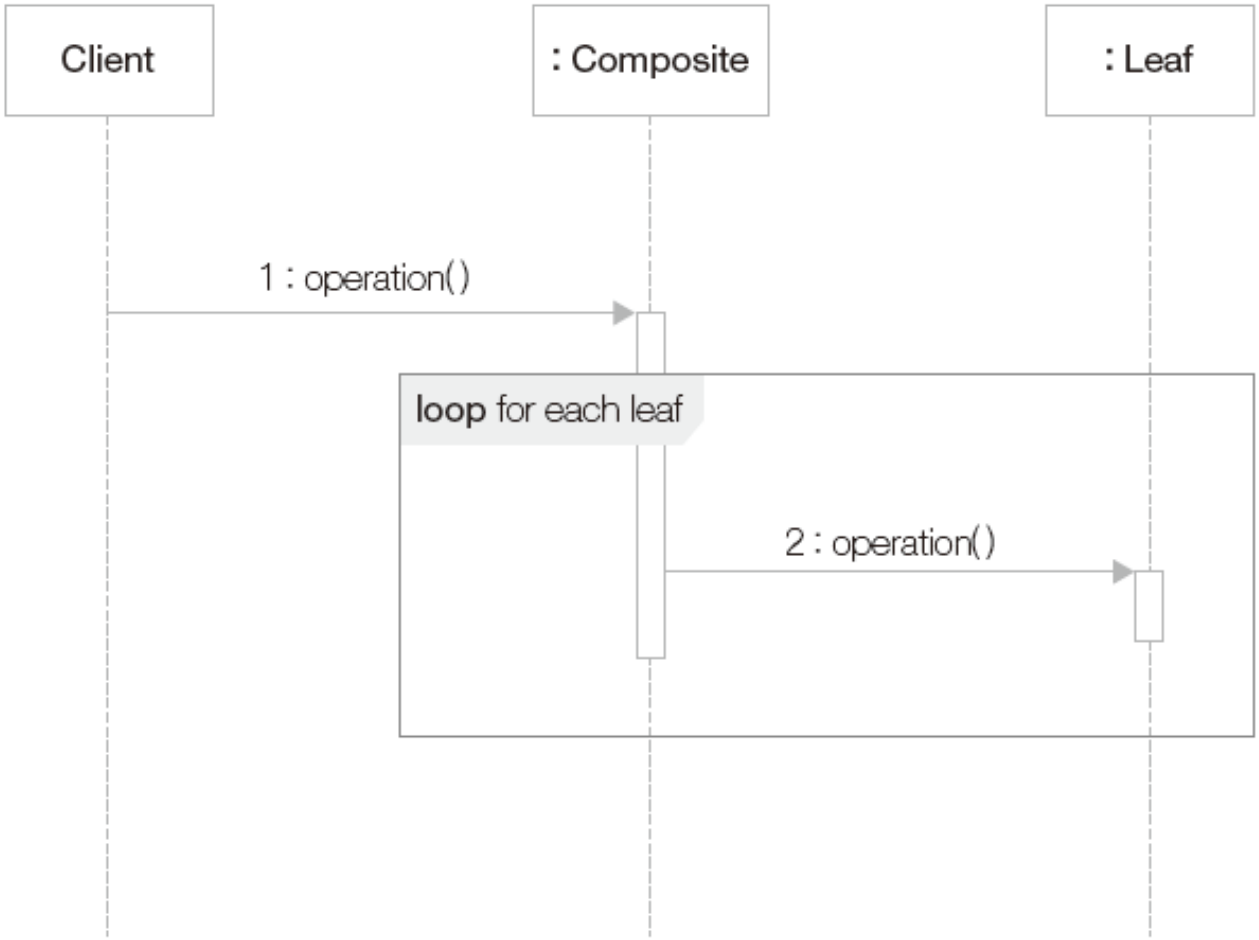
14.4 컴퍼지트 패턴

그림 14-7 컴퍼지트 패턴의 컬레보레이션



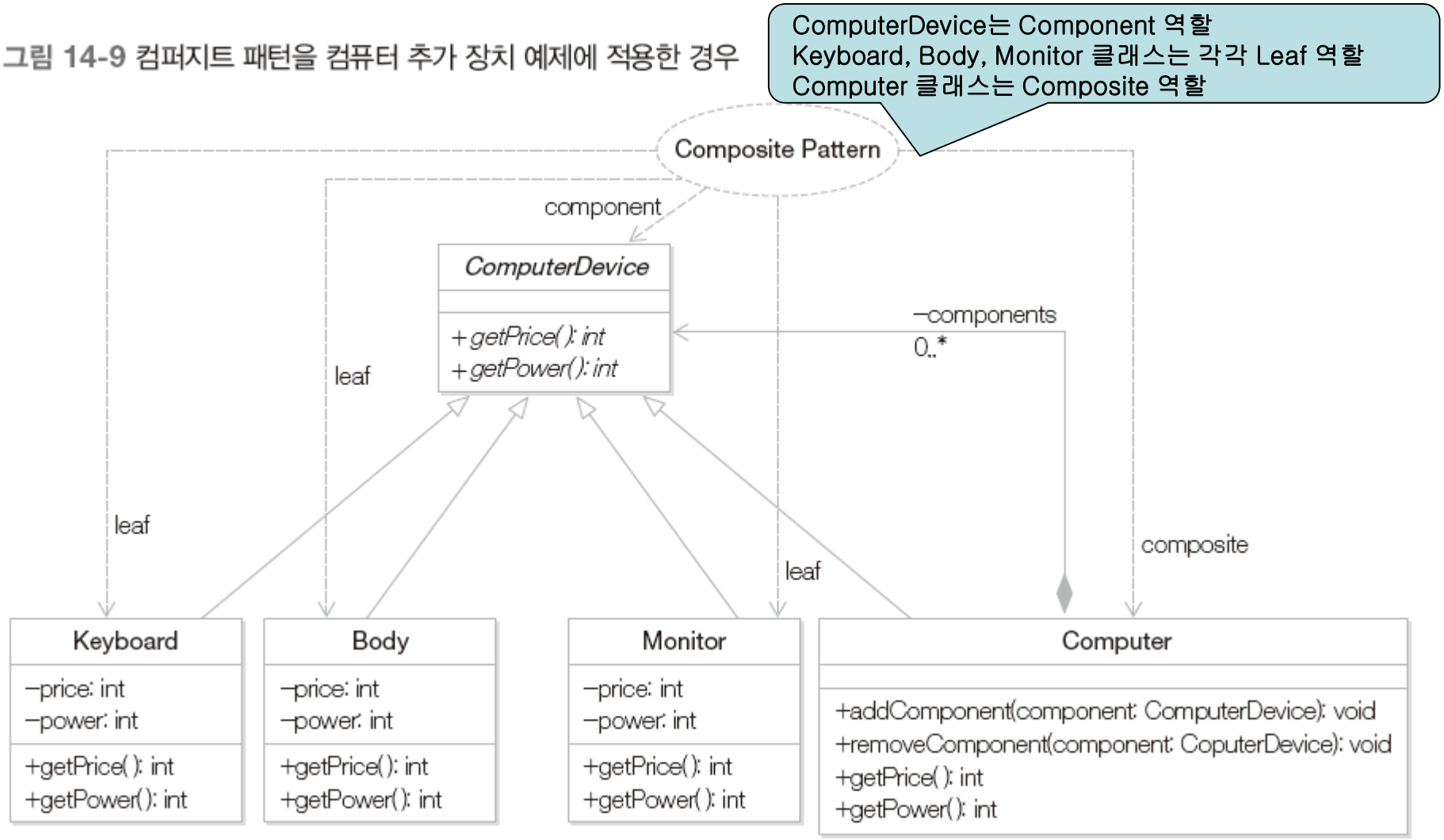
14.4 컴퍼지트 패턴

그림 14-8 컴퍼지트 패턴의 순차 다이어그램



컴퍼지트 패턴의 적용

그림 14-9 컴퍼지트 패턴을 컴퓨터 추가 장치 예제에 적용한 경우



게임 상점

- 게임의 상점 기능 구현
  - 인벤토리가 있으며, 인벤토리에는 아이템을 넣을 수 있음
  - 가방이라는 아이템은 인벤토리 효율을 높여주고, 가방 안에 여러 개의 아이템 보관이 가능
    - 가방은 깊이 제한 없이 계속 중첩될 수 있는 트리 형태로 만들어짐

- 아이템은 이름, 가격을 가지고 있음

```
public interface Item {
    int getPrice();
    String getName();
}
```

*출처 : 컴포지트 패턴(Composite Pattern, 컴포짓 패턴) 이란?, https://jake-seo-dev.tistory.com/399

게임 상점

- DefaultItem 클래스

```
public class DefaultItem implements Item {
    private final int price;
    private final String name;

    public DefaultItem(int price, String name) {
        this.price = price;
        this.name = name;
    }

    @Override
    public int getPrice() {
        return price;
    }

    @Override
    public String getName() {
        return name;
    }
}
```

게임 상점

- ItemStorage 인터페이스
  - 아이템을 저장하거나 뺄 수 있음
  - 모든 아이템의 가격을 구할 수 있는 메서드 포함

```
public interface ItemStorage {
    void addItem(Item item);
    void removeItem(Item item);
    int getAllPrice();
}
```

게임 상점

- DefaultItemStorage 구현

```
public abstract class DefaultItemStorage implements ItemStorage{
    private ArrayList<Item> items = new ArrayList<>();

    @Override
    public void addItem(Item item) {
        items.add(item);
    }

    @Override
    public void removeItem(Item item) {
        items.remove(item);
    }

    @Override
    public int getAllPrice() {
        return items.stream().mapToInt(Item::getPrice).sum();
    }
}
```

게임 상점

- Inventory 구현
  - 인벤토리는 DefaultItemStorage로 충분하여 상속만 받음

```
public class Inventory extends DefaultItemStorage {
}
```

게임 상점

❖ ItemBag 구현

- ItemBag은 Item 속성과 ItemStorage 속성이 모두 필요

```
public class ItemBag extends DefaultItemStorage implements Item {
    private final String name;
    private final int price;

    public ItemBag(int price, String name) {
        this.name = name;
        this.price = price;
    }

    @Override
    public int getPrice() {
        return this.getAllPrice() + this.price;
    }

    @Override
    public String getName() {
        return this.name;
    }
}
```

게임 상점

❖ Client 코드

```
public class Client {
    public static void main(String[] args) {
        Inventory inventory = new Inventory();
        Item longSword = new DefaultItem(350, "긴 검");
        inventory.addItem(longSword);

        ItemBag beginnerBag = new ItemBag(100, "모험자의 가방");
        Item rareSword = new DefaultItem(400, "레어 검");
        Item uniqueSword = new DefaultItem(1000, "유니크 검");
        beginnerBag.addItem(rareSword);
        beginnerBag.addItem(uniqueSword);
        inventory.addItem(beginnerBag);

        System.out.println("롱소드의 가격: " + getPrice(longSword)); // 롱소드의 가격: 350
        System.out.println("모험자의 가방과 내부 아이템들의 가격: " + getPrice(beginnerBag));
        System.out.println("인벤토리 아이템 가격의 총 합계: " + inventory.getAllPrice());
    }

    public static int getPrice(Item item) {
        return item.getPrice();
    }
}
```

컴퍼지트 패턴의 장점

1. **유연성**: 개별 객체와 복합 객체를 동일하게 취급할 수 있음.
2. **확장성**: 새로운 Leaf나 Composite를 추가해도 기존 코드에 영향을 미치지 않음.
3. **단순화**: 클라이언트는 객체의 구체적인 타입이나 구조에 신경 쓸 필요가 없음.

컴퍼지트 패턴의 단점

1. **복잡성 증가**: 트리 구조를 설계하고 관리하는 데 추가적인 비용이 발생할 수 있음.
2. **오버헤드**: 구조가 지나치게 복잡해지면 관리와 디버깅이 어려워질 수 있음.

컴퍼지트 패턴 사용 사례

1. **그래픽 UI 구성**: 버튼, 텍스트 필드, 패널 등을 트리 구조로 관리.
2. **파일 시스템**: 파일(Leaf)과 디렉터리(Composite)를 관리.
3. **조직도**: 직원(Leaf)과 부서(Composite)를 계층적으로 표현.

❖ 컴퍼지트 패턴은 복잡한 계층적 구조를 효과적으로 관리하고 클라이언트 코드의 단순화를 돕는 데 적합함