고급 객체지향 개발론

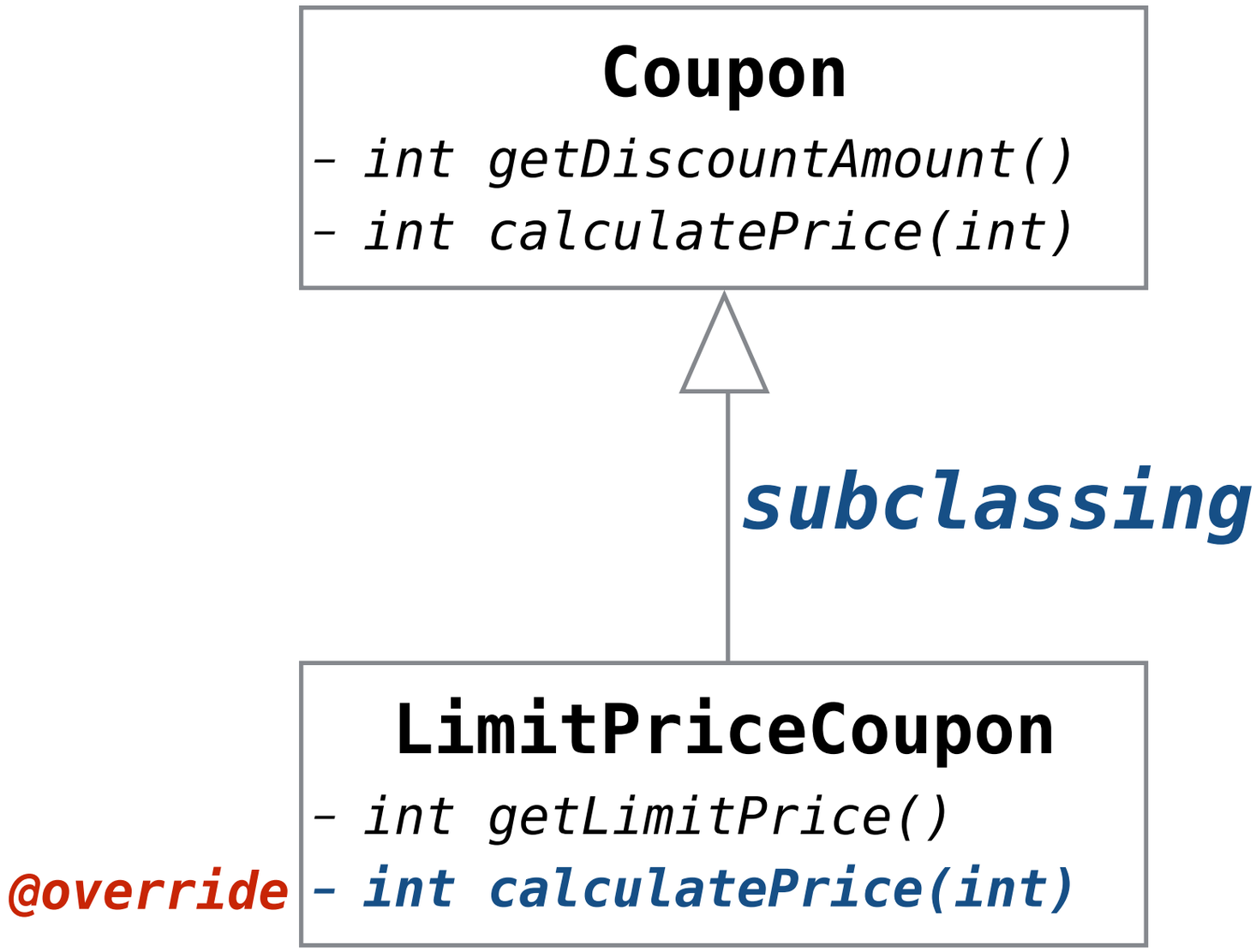
05.

다형성과 추상타입
재사용: 상속보단 조립

daumkakao
최윤상

상속

inheritance


Coupon
- int getDiscountAmount()
- int calculatePrice(int)
subclassing
LimitPriceCoupon
- int getLimitPrice()
@override
- int calculatePrice(int)

```
Coupon coupon = new LimitPriceCoupon(…);
// apply coupon.
int price =
coupon.calculatePrice(procut.getPrice());
```

which one?
Coupon#calcuatePrice(int) ?
or
LimitPriceCoupon#calculatePrice(int) ?

다형성

polymorphism

다형-성 (多形性) | -썽 |

명사 [생물학, 생리학]

동일종(同一種)의 생물이면서도 형태나 성질이 다양성을 보이는 상태. 암수에 의한 크기·형태·색깔 등의 차이와 꿀벌에서의 여왕벌과 일벌 같은 것.

다형성

English

프로그램 언어의 **다형성**(polymorphism; 폴리모피즘)은 그 프로그래밍 언어의 자료형 체계의 성질을 나타내는 것으로, 프로그램 언어의 각 요소들(상수, 변수, 식, 오브젝트, 함수, 메소드 등)이 다양한 자료형(type)에 속하는 것이 허가되는 성질을 가리킨다. 반댓말은 **단형성**(monomorphism)으로, 프로그램 언어의 각 요소가 한가지 형태만 가지는 성질을 가리킨다.

Type of Polymorphism

Ad-hoc polymorphism

Parametric polymorphism

Subtyping

Type of Polymorphism

Ad-hoc polymorphism

Parametric polymorphism

Subtyping

Ad-hoc polymorphism

function overloading

```
program Adhoc;

function Add( x, y : Integer ) : Integer;
begin
    Add := x + y
end;

function Add( s, t : String ) : String;
begin
    Add := Concat( s, t )
end;

begin
    Writeln(Add(1, 2));                    (* Prints "3"              *)
    Writeln(Add('Hello, ', 'World!'));     (* Prints "Hello, World!" *)
end.
```

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

Parametric polymorphism

generics

```
class List<T> {
    class Node<T> {
        T elem;
        Node<T> next;
    }
    Node<T> head;
    int length() { ... }
}

List<B> map(Func<A,B> f, List<A> xs) {
    ...
}
```

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

Subtyping

subclassing(inheritance)

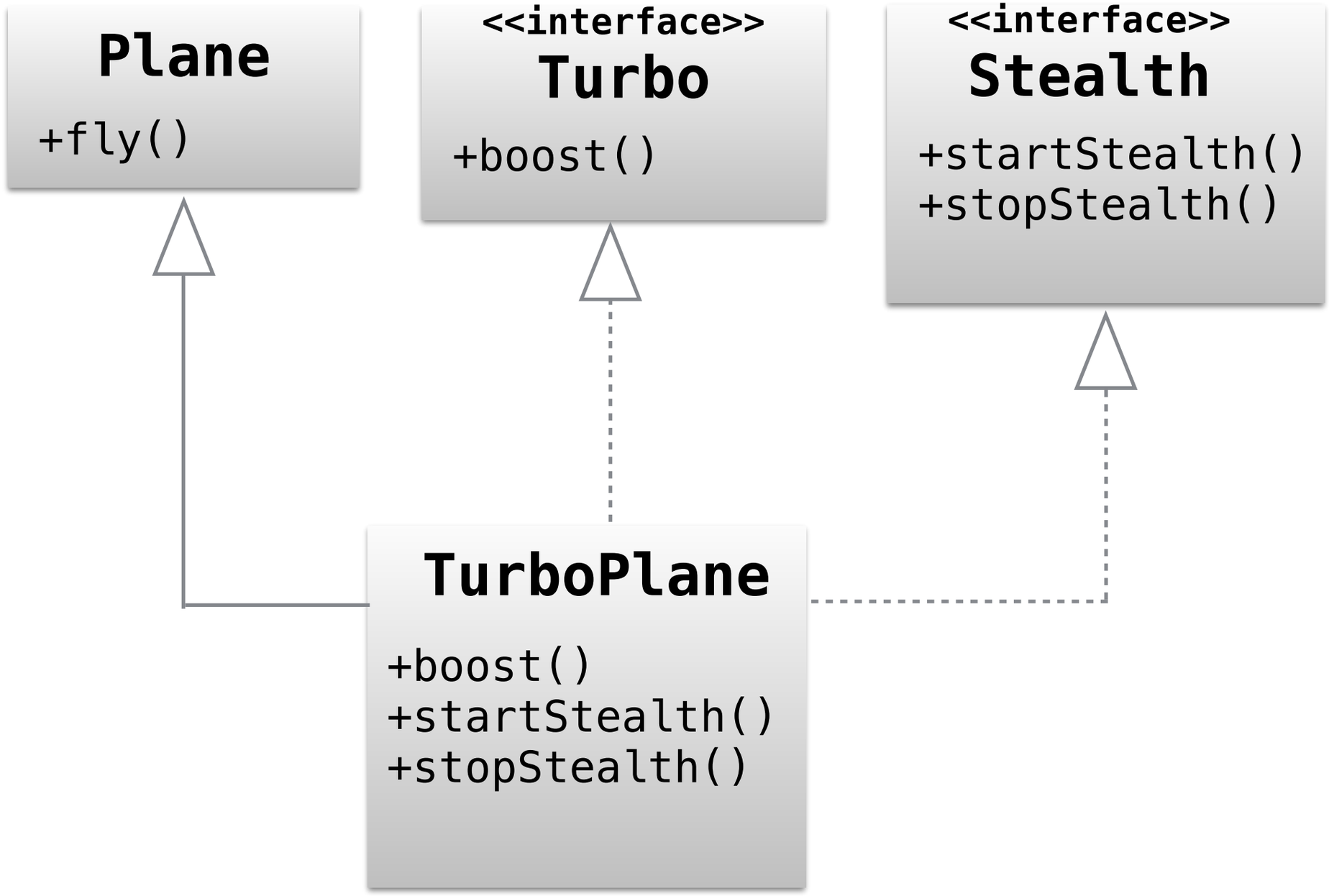
```
abstract class Animal {
    abstract String talk();
}

class Cat extends Animal {
    String talk() {
        return "Meow!";
    }
}

class Dog extends Animal {
    String talk() {
        return "Woof!";
    }
}
```

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

Subtyping

인터페이스 상속 *inheritance-interface*

구현 상속 *inheritance implementation*

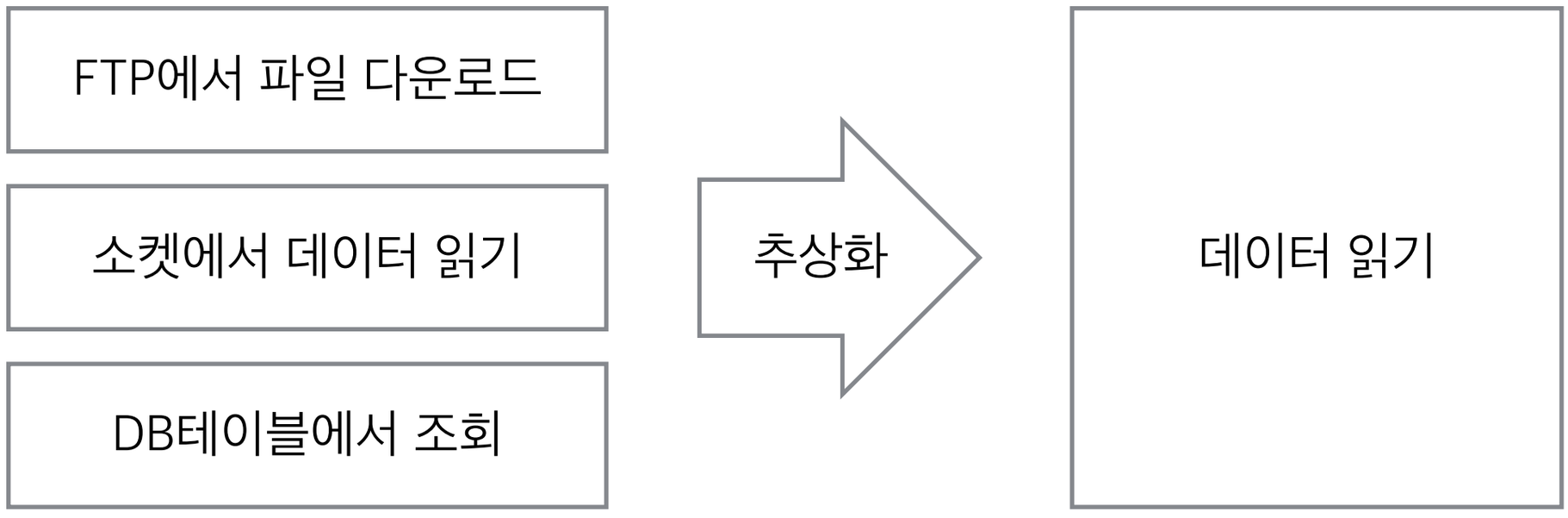

Plane
+fly()
<<interface>>
Turbo
+boost()
<<interface>>
Stealth
+startStealth()
+stopStealth()
TurboPlane
+boost()
+startStealth()
+stopStealth()

```
Plane plane = new TurboPlane();
plane.fly(); // ok.
plane.boost(); // compile error!
plane.startStealth(); // compile error!
```

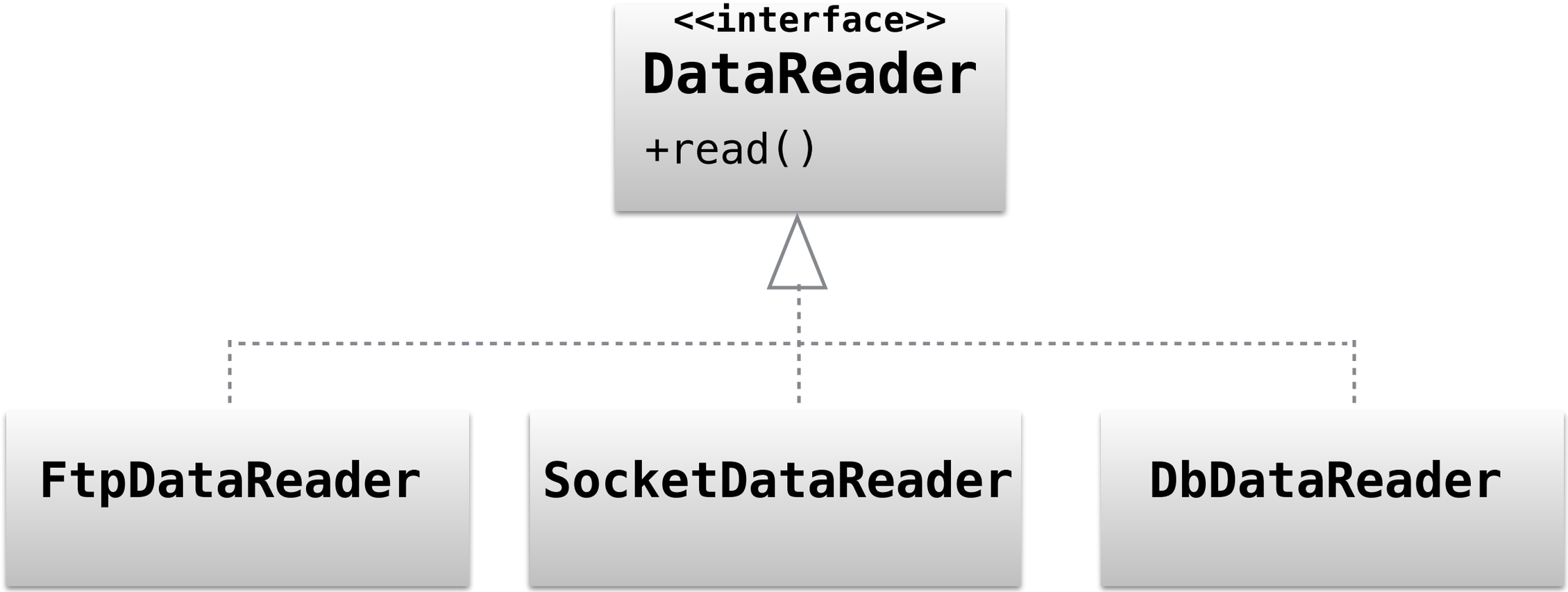
```
interface TurboAssistor {
   void autoBoost(Turbo turbo);
   …
}
```

```
TurboPlane p = new TurboPlane();
TurboAssistor a = …;
a.autoBoost(p);
```

추상타입



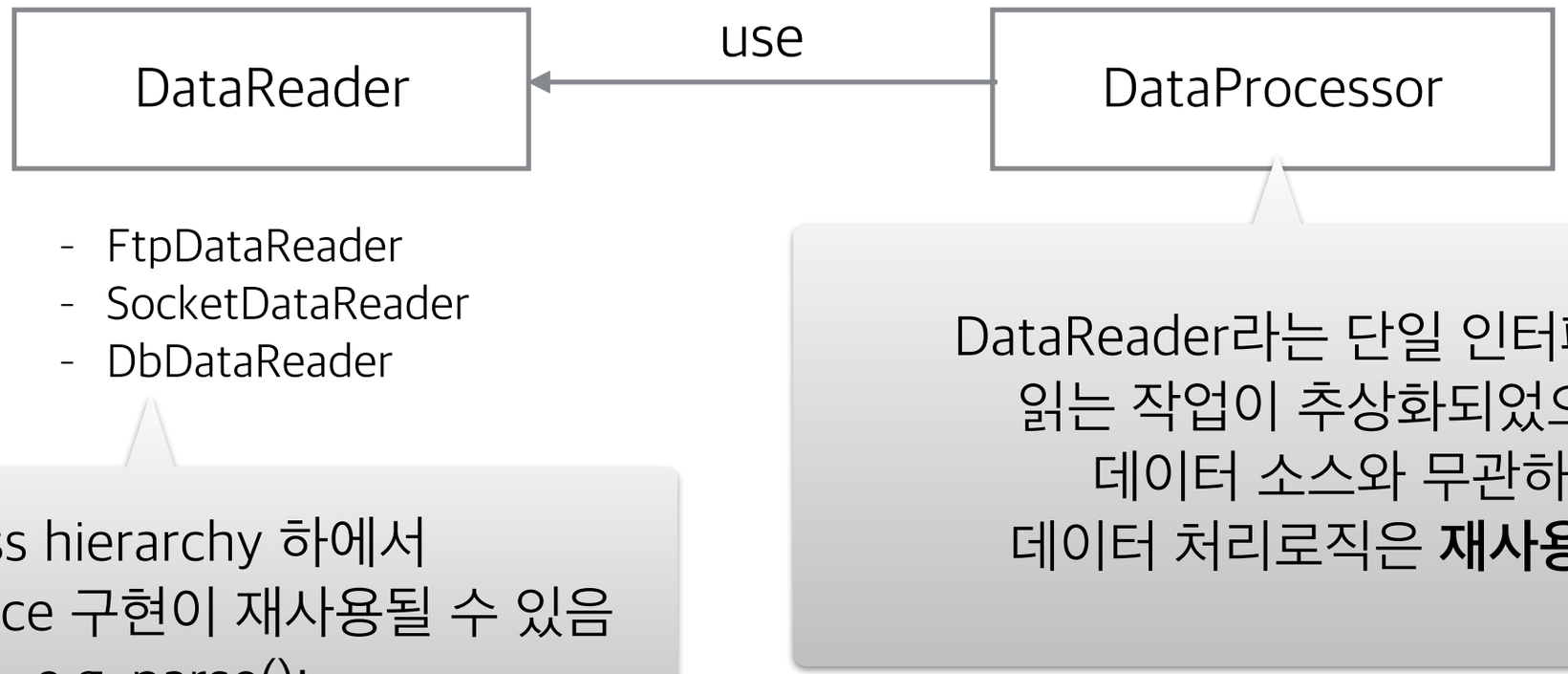
추상타입


<<interface>>
DataReader
+read()
FtpDataReader
SocketDataReader
DbDataReader

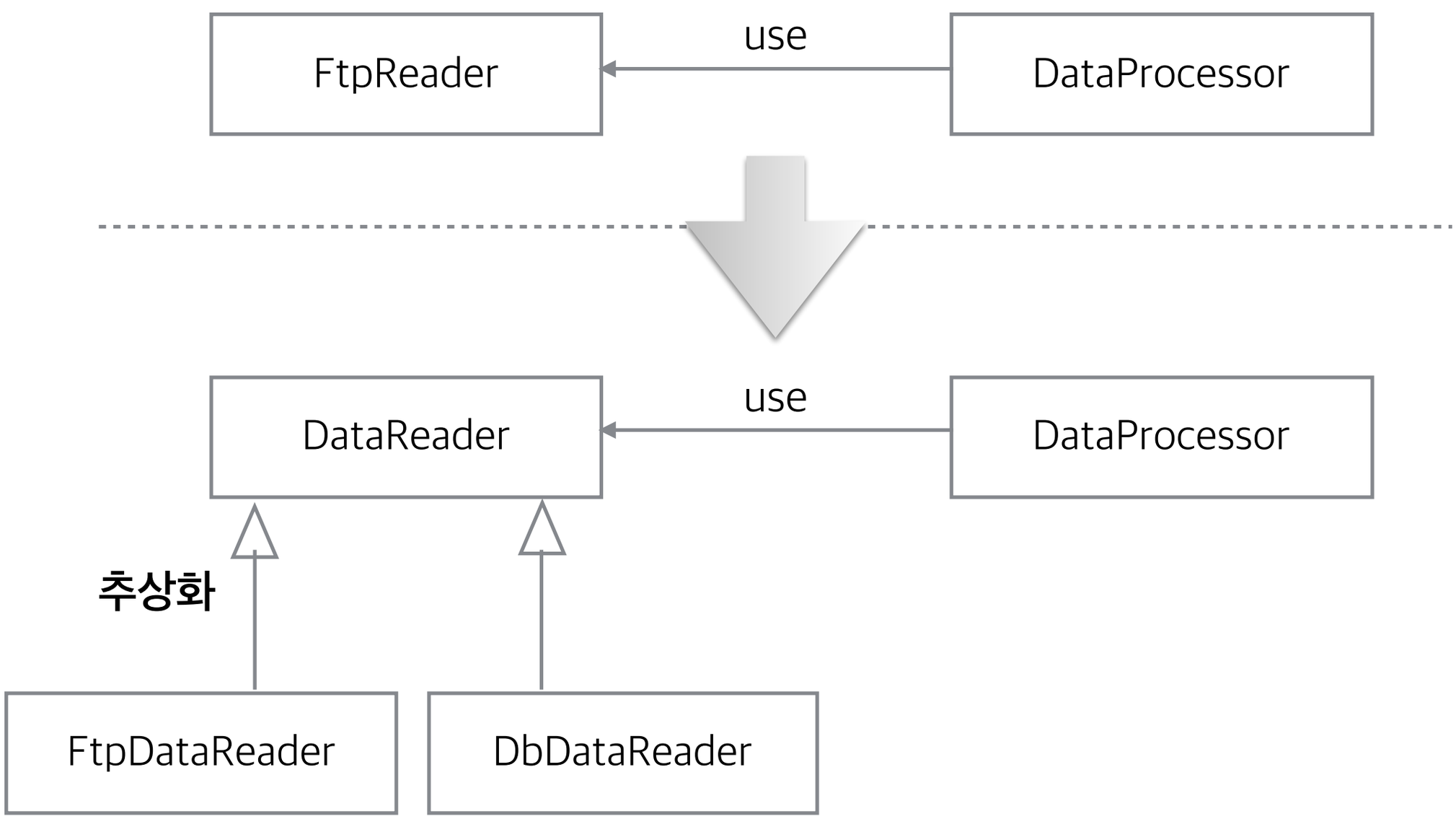
```
class DataProcessor {
  public void setDataReader(DataReader r);
  public void proccess();
  …
}
```

```
DataProcessor processor = …;
DataReader reader = new FtpDataReader(…);
processor.setDataReader(reader);
processor.process();
```

추상타입에 의한 재사용

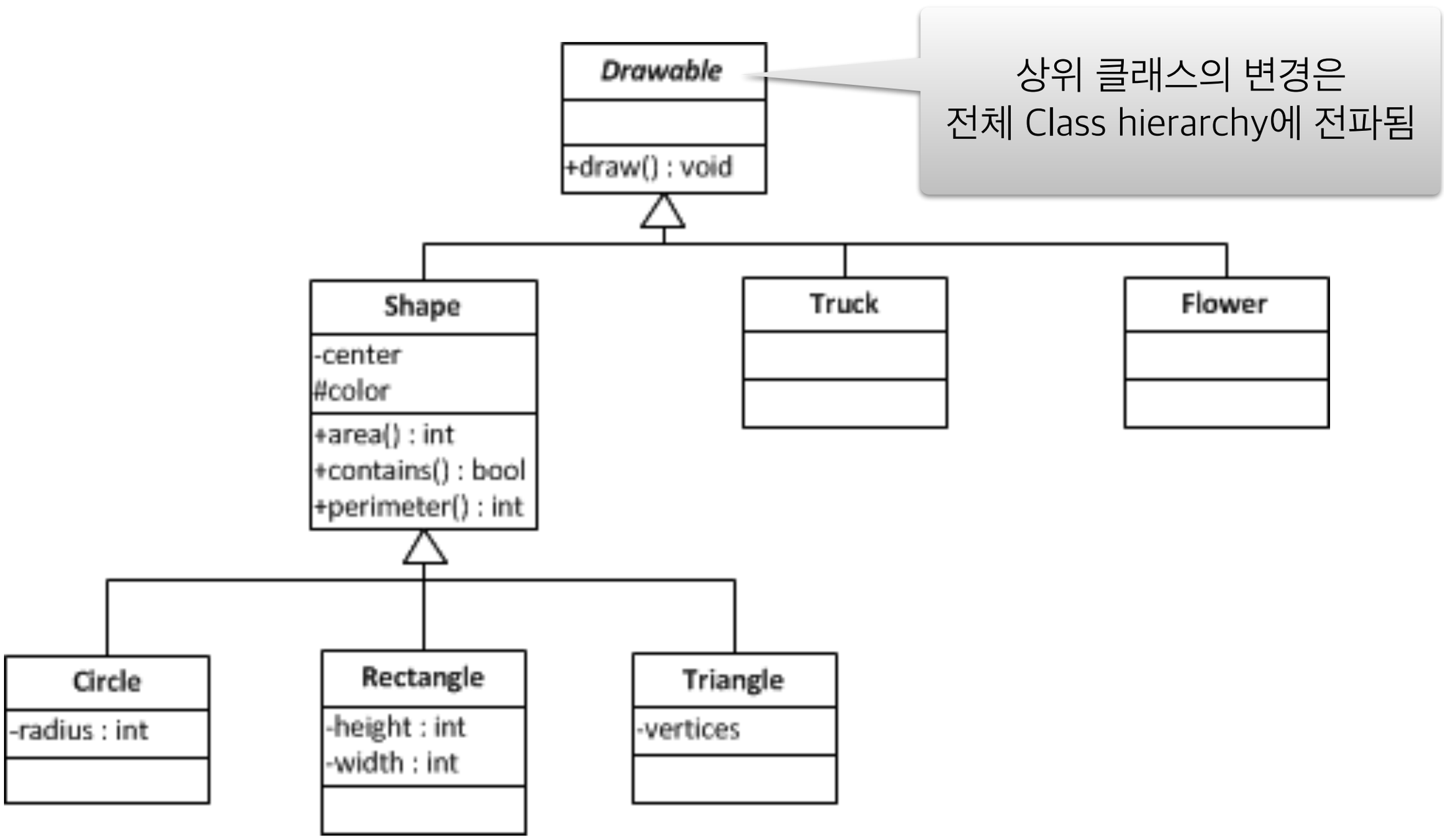

DataReader
use
DataProcessor
- FtpDataReader
- SocketDataReader
- DbDataReader
class hierarchy 하에서
특정 interface 구현이 재사용될 수 있음
e.g. parse();
DataReader라는 단일 인터페이스로
읽는 작업이 추상화되었으므로
데이터 소스와 무관하게
데이터 처리로직은 재사용 가능

접근방법: 확장이 필요할 때

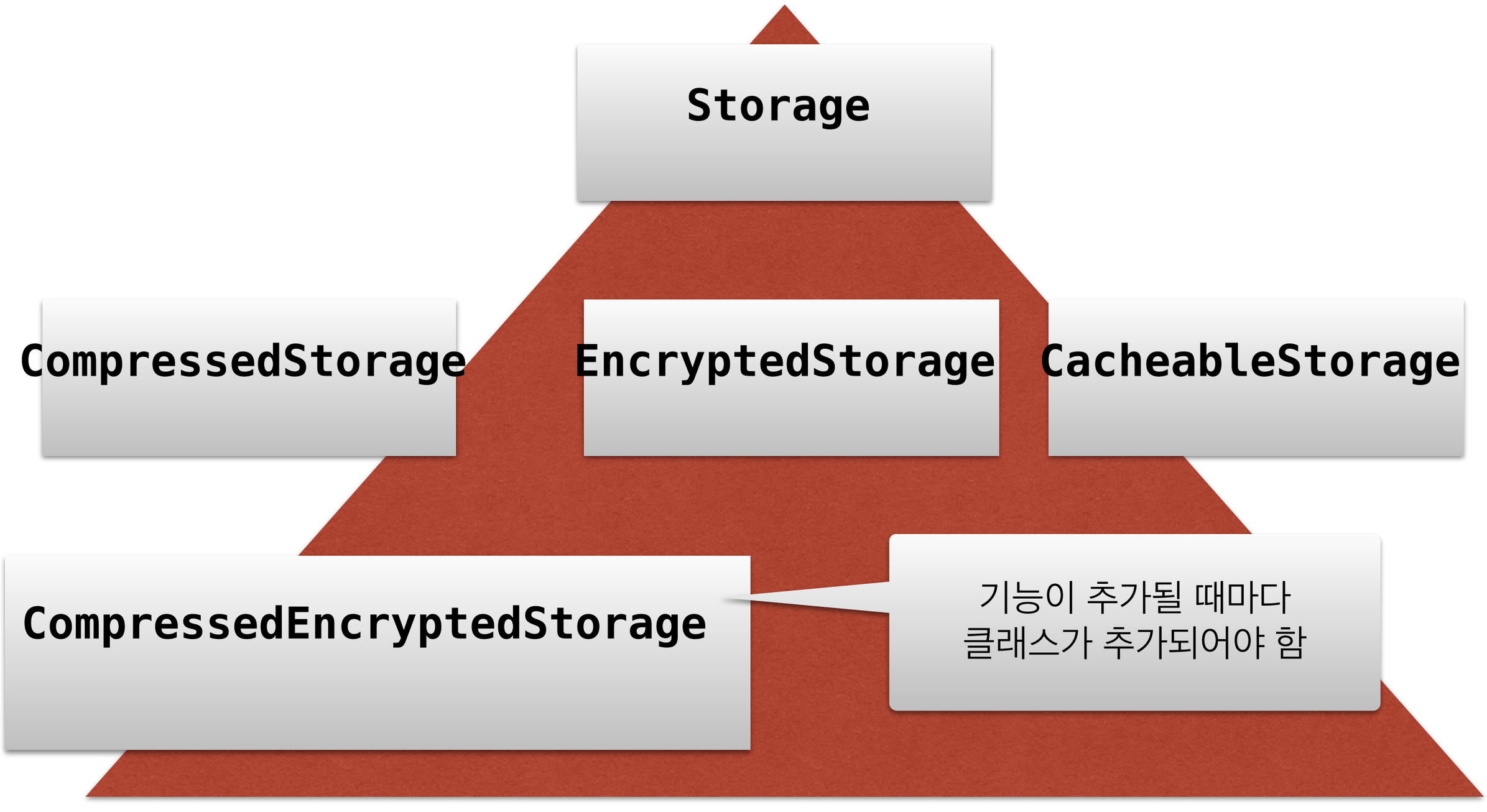


상속을 통한 재사용의 단점

#1 상위 클래스 변경이 어려움



#2 클래스의 불필요한 증가



#3 상속의 오용

```
public class Container
        extends ArrayList<Luggage> {
    ......
}
```

Container “IS-A”? ArrayList

조립을 이용한 재사용

단위 기능 별 클래스
+
기능을 조립하는 클래스

Storage

Compressor

Encryptor

CacheEngine

상속보다는 조립

상속의 문제를 대부분 회피할 수 있음

동적으로 기능을 변경할 수 있음

상속은 언제 써야 하나?

재사용 관점이 아니라 확장의 관점에서

“저 클래스의 xx()메소드가 탐나니 상속받아 쓰자” (x)

“IS-A” 관계가 성립하는 경우만

명확한 계층구조가 있는 경우