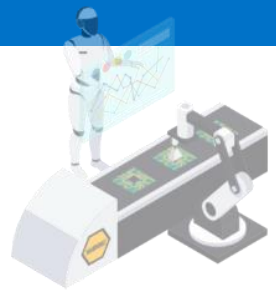
13강. 클래스 다이어그램과 객체 다이어그램

컴퓨터과학과 김희천 교수

목차



Chapter. 1

객체지향 개념과 클래스

1. 클래스와 클래스 다이어그램

✚ 클래스는 객체의 설계도, 객체는 클래스의 인스턴스

- 클래스는 객체에 대한 설계를 제공
- 특정 자동차 모델(또는 공장)은 클래스, 홍길동의 자동차는 객체
- 클래스로부터 만들어진 객체들은 시스템 동작 중에 생성되어 메시지를 주고받으며 소멸됨

✕ 클래스 다이어그램

- 클래스의 명세, 그리고 클래스 간의 관계를 보여줌
- 실제 동작에 필요한 구성 요소들과 이들의 관계를 보여줌
- UML에서 가장 활용도가 높은 다이어그램
- 분석 과정에서 개념 클래스는 설계 과정에서 구현 클래스로 구체화됨

2. 클래스의 구성

상태

- 객체가 가지는 정보(데이터 값)
- 실제 구현에서는 객체에 소속된 변수(속성)로 나타남
- 예) 자동차의 경우 차량 번호, 색상, 연식, 소유자, 주행 거리 등

동작

- 객체가 수행할 수 있는 기능
- 실제 구현에서는 메소드로 표현됨
- 예) 자동차의 경우 기어 변속, 가속, 정지, 문 잠그기, 시동 걸기 등

3. 객체지향 개념

추상화

- 복잡한 것을 간단히 표현하는 것
- 클래스의 설계 목적에 맞는 속성과 메소드를 선택하고 불필요한 것을 제거함
- 분석이나 설계에서 나오는 모델이란 용어는 추상화된 표현을 의미

캡슐화

- 캡슐에 담아 상호작용에 필요한 것만 노출하는 것
- 노출 정보를 제외한 내부의 속성과 메소드를 숨김
- 인터페이스가 그대로 유지되면 내부에 변경이 있더라도 외부에 영향을 주지 않음
- 정보 은닉과 관계 있음

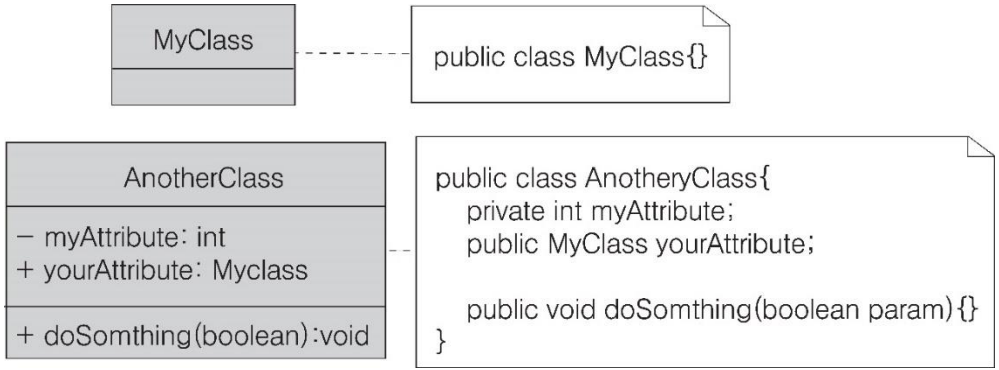
Chapter. 2

클래스 표기

1. 클래스의 작성(1/4)

✚ 클래스 표기법

- ✕사각형으로 표현하며 3개 영역으로 나뉨
- ✕위에서부터 순서대로 클래스의 이름, 속성, 메소드를 표시
- ✕속성이나 메소드는 생략할 수 있음
- ✕속성과 메소드의 가시성을 표현해 줄 수 있으며, 노트를 사용하여 자세한 설명을 추가할 수 있음



1. 클래스의 작성(2/4)

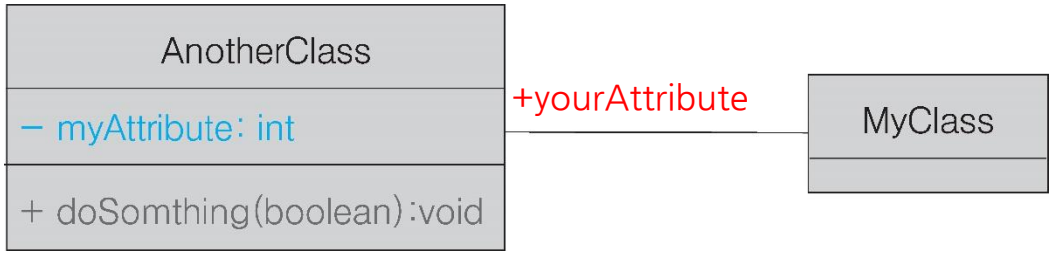
✚ 속성과 메소드의 접근 제어(가시성)

접근제어	표기	설 명
public	+	클래스 외부에서 접근이 가능
protected	#	클래스 내부와 동일 패키지 또는 상속 받은 클래스에서 접근 가능
package	~	클래스 내부 또는 동일한 패키지에서 접근 가능
private	-	클래스 내부에서만 접근이 가능

1. 클래스의 작성(3/4)

✚ 속성의 표시

- ✕ 클래스 기호의 두 번째 영역 안에 표기함
- ✕ 연관을 이용하여 속성을 표시해 줄 수도 있음.
  다이어그램이 복잡해지나 속성의 유형을 명확히 보여줄 수 있음
- ✕ 예) AnotherClass에서 속성으로 MyClass 유형의 yourAttribute를 가질 때



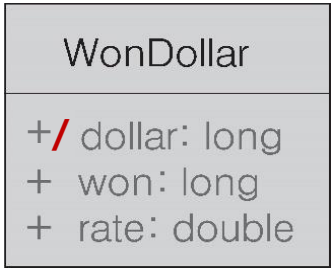
1. 클래스의 작성(4/4)

✚ 속성의 형식

- 접근제어 / 이름 : 타입 관계수 = 기본값 {제약사항}
- 예: - orders : Order[1..10]=null {ordered}
- '/'는 유도된 속성을 의미
- 이름은 반드시 필요하며, 속성의 타입은 구현 언어에 종속적
- 관계수는 배열과 같은 집합체임을 표시하기 위한 것
- '=0' 이나 '=null'과 같이 기본값을 지정해 줄 수 있음

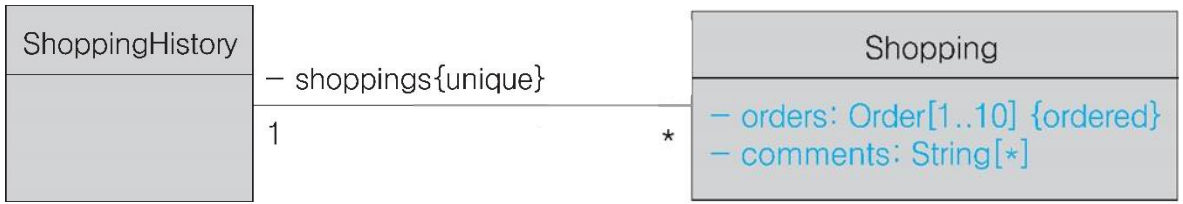
✚ 유도된 속성

- 다른 속성들로부터 값이 얻어지는 속성
- 유도된 속성은 설계 단계에서 나타나며 구현 단계에서는 읽기 메소드로 변환됨



2. 관계수

- 배열과 같이 1개 이상의 요소(또는 객체)들로 구성된 속성을 표현
  - 예) [1..10] 또는 [*]
- 연관을 사용하여 속성을 표현하는 경우는 연관의 양쪽 끝에 관계수를 표시함
  - 예) 1개의 ShoppingHistory는 0개 이상의 Shopping 객체(shoppings)를 가짐

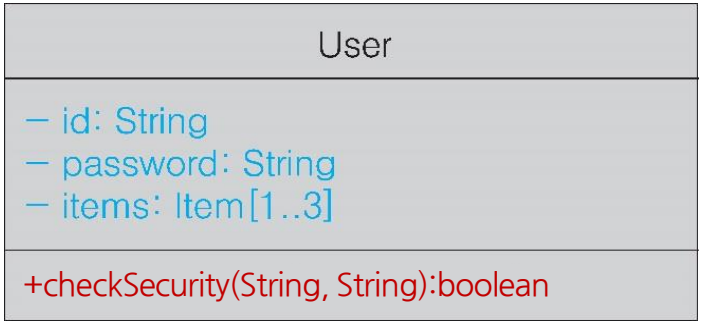


3. 제약 사항

- 제약 조건으로 지켜야 하는 규칙을 의미
  - 속성의 성질을 상세하게 명시할 수 있음
  - 중괄호 { … } 안에 텍스트로 표시함
- 예
  - {unique} – 중복이 없어야 함
  - {ordered} - 객체들이 정렬되어 있어야 함
  - {readOnly} - 속성이 상수임
  - {ordered, unique} - 순서 집합
  - {ordered, not unique} - 시퀀스
  - {not ordered, unique} - 집합
  - {not ordered, not unique} - 중복 집합

4. 메소드

- 클래스 기호의 세 번째 영역 안에 표기함
- 메소드의 형식
  - 접근제어 이름(파라미터들):타입 {제약사항}
- 파라미터의 형식
  - 방향 이름:타입 관계수 =기본값 {제약사항}
  - 파라미터가 여럿 있으면 컴마(,)로 구분함
  - 방향은 in/out의 구분으로 입력값인지 출력값인지를 구분하기 위한 것
  - 메소드의 시그너처에서 파라미터의 이름은 중요하지 않음



5. 정적 요소

- 속성이나 메소드를 정적(static)으로 선언할 수 있음
  - 모든 객체가 공유하는 속성이나 메소드를 의미
  - 특정 객체에 속한 것이 아님
  - 정적 속성과 메소드는 클래스 정의 영역에 속함
  - 예) 특정 클래스에서 현재 생성되어 있는 객체의 수를 저장하는 변수
- 속성이나 메소드의 이름에 밑줄을 그어 표시함

Chapter. 3

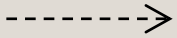
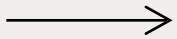
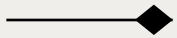
클래스 다이어그램

1. 클래스 다이어그램 개요

- **개별 클래스 또는 인터페이스의 정의에 관한 명세**
  - 이름, 속성과 메소드 정의를 포함
- **클래스 간의 관계를 표현**
  - 의존, 연관, 상속, 집합체 연관(aggregation), 구성 집합체 연관(composition) 등
- **템플릿의 사용**
  - 일반적 클래스로서 재사용성을 높이기 위한 설계
- **제약 조건의 사용**
  - OCL은 UML 요소의 제약 조건을 명시하기 위한 언어
  - 중괄호 { }나 노트를 사용하여 표시됨

2. 클래스 간의 관계

✚ 클래스 간의 결합 정도에 따른 관계의 분류

표기	이름	설 명
	의존	한 클래스가 다른 클래스와 최소한의 사용 관계가 발생한 경우
	연관	한 클래스가 다른 클래스와 지속적인 관계를 맺게 되는 경우
	집합체 연관	한 클래스가 다른 클래스의 객체에 대한 레퍼런스를 공유하는 경우
	구성 집합체 연관	한 클래스가 다른 클래스의 객체를 포함하는 경우
	일반화(상속)	한 클래스가 다른 클래스를 상속받는 경우

3. 의존 관계

- 단순한 사용이나 참조를 의미하며 가장 약한 결합으로 일시적 사용 관계
- 클래스 A가 클래스 B의 객체를 사용하는 경우 점선의 화살표로 표시
  - A가 B를 사용하려면 B에 대하여 알아야 함
  - Java에서 클래스가 A가 클래스 B를 import 하여 사용하는 경우
  - 클래스 A에서 메소드의 파라미터나 지역 변수로 클래스 B를 사용하는 경우
  - A에서 유틸리티를 제공하는 B를 사용하는 경우

4. 연관

- 클래스 A의 속성으로 클래스 B의 객체를 사용하는 경우
- 실선 또는 실선의 화살표로 표시
  - 화살표가 없다면 양방향을 의미
  - 연관의 이름과 역할을 표시
- 연관의 구현 예


```
class BlogAccount
{
  // BlogEntry와의 연관으로 인한 속성
  private BlogEntry[] entries;
}
class BlogEntry
{
  // BlogAccount와의 연관으로 인한 속성
  private BlogAccount blog;
}
```

5. 집합체 연관

✚ 연관의 강한 형태로, 전체와 부품의 포함 관계
- 전체 쪽에 속이 빈 마름모를 두고 선으로 연결하여 표시
- 공유 집합체 연관(aggregation)이라고도 함

✚ 예를 들어 클래스 A(전체)가 클래스 B(부품)의 객체를 포함하고 있는 경우
- 클래스 A의 정의에서 B 객체의 참조값을 속성으로 가짐
- 부품은 다른 전체에도 포함될 수 있음.
  즉 부품이 2개 이상의 전체에 의해 공유될 수 있음
  (B 객체는 다른 전체 객체에도 포함될 수 있음)

✕ 예

- A Rectangle has 4 Segments.
  각 Segment는 0개 이상 Rectangle의 부품일 수 있다

6. 구성 집합체 연관

✚ 집합체 연관보다 강한 포함 관계

- 전체와 부품은 생명주기가 같음
- 전체가 삭제되면 부품이 같이 삭제됨
- 부품은 공유되지 않음
- 전체 쪽에 검은 마름모를 두고 선으로 연결하여 표시

✕ 예

- 한 Folder는 0개 이상의 File을 포함할 수 있다.
  각 File은 정확히 1개의 Folder에 포함된다.
  Folder가 삭제되면 포함된 모든 File도 삭제 된다.

7. 일반화

- 부모와 자식 사이의 상속 관계로 특정 클래스(자식)가 다른 클래스(부모)의 한 종류임을 나타냄
  - 자식 클래스를 파생 클래스 또는 서브 클래스라 하며, 부모 클래스를 기반 클래스 또는 슈퍼 클래스라고도 함
- 자식 클래스(A)는 부모 클래스(B) 의 모든 속성과 메소드를 상속 받음
  - 부모로 향하는 화살표로 표시(부모 클래스 쪽에 속이 빈 삼각형 머리)
  - B는 A를 일반화 한 것, A는 B의 한 종류(A is a kind of B)
  - 예) Manager is a kind of Employee

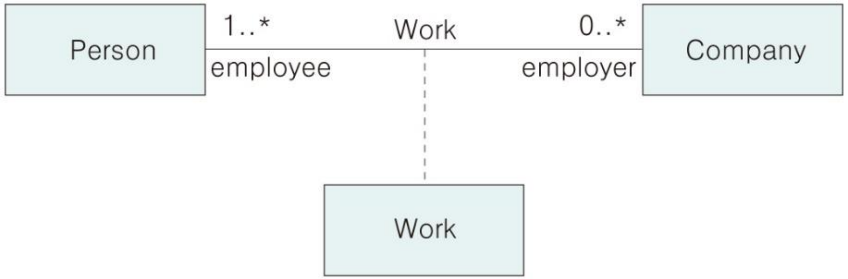
8. 클래스 관계에 대한 고찰(1/2)

+ 일반화와 재사용 관계

- × 자식 클래스는 부모가 가진 속성과 메소드를 상속받기 때문에 상속 관계는 다른 관계에 비해 강한 결합
- × 단순히 코드를 재사용하기 위해 상속 관계를 만들지 말 것
- × 분명히 의미적으로 부모의 한 종류일 때만 자식 클래스로 만들 것

+ 연관 클래스

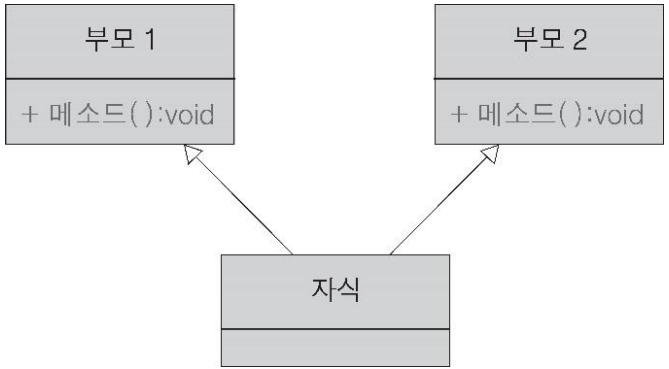
- × 중요한 연관을 독립적 클래스로 표현
- × 속성(연관의 특성)과 메소드를 가짐
- × 연관클래스를 연관과 점선으로 연결



8. 클래스 관계에 대한 고찰(2/2)

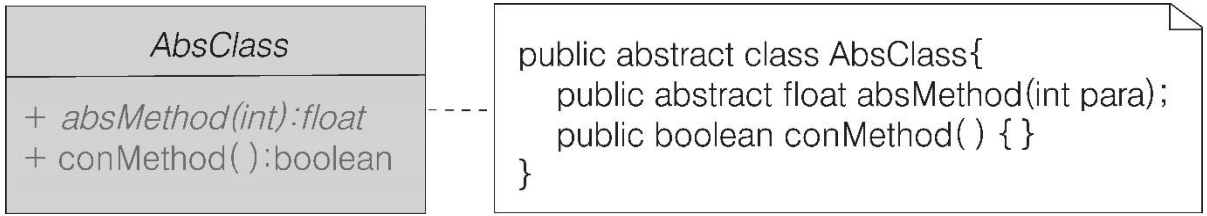
✚ 다중 상속

- ✕자식 클래스가 2개 이상의 부모로부터 상속받는 경우
- ✕동일한 이름의 속성이나 메소드를 상속받으면 문제가 발생
- ✕Java에서는 다중 클래스 상속을 할 수 없으며 다중 인터페이스 상속은 가능함



9. 추상 클래스

- 클래스를 설계하면서 특정 메소드를 정의하고 싶지 않거나, 정의할 수 없을 때 메소드의 시그너처만을 표시함
- 추상 메소드를 포함하는 클래스는 추상 클래스
  - 구현이 없는 메소드를 추상 메소드라 함
  - 추상 클래스로부터 직접적으로 객체를 생성할 수 없음
  - 자식 클래스에서 추상 메소드를 구현해 주면 자식 클래스의 객체를 생성할 수 있음
- UML에서는 이탤릭체를 사용하여 *추상 메소드* 나 *추상 클래스*를 표현함



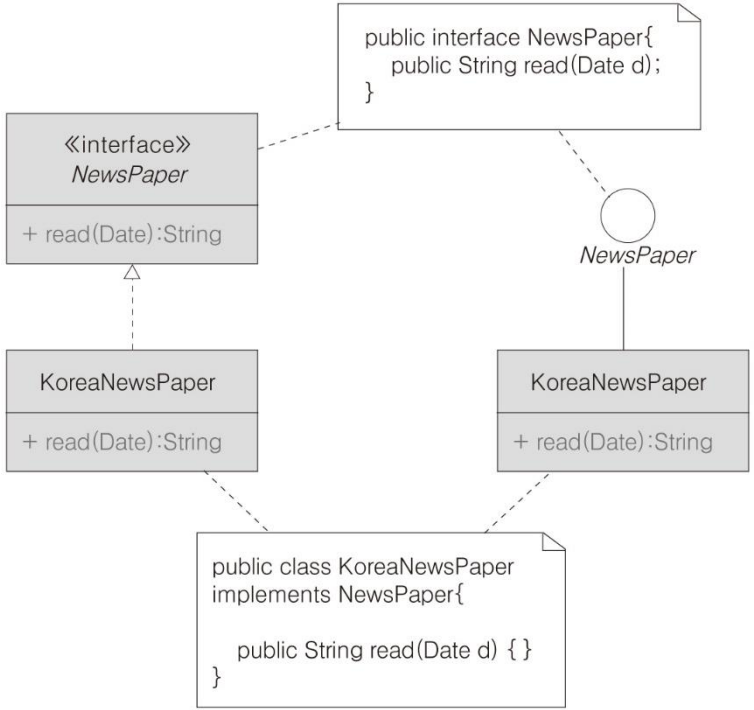
10. 인터페이스 (1/2)

- 인터페이스는 모든 메소드가 추상 메소드인 클래스
  - 추상 클래스와 마찬가지로 인터페이스로부터 직접 객체를 생성할 수 없음
  - 자식 클래스들은 동일한 서비스를 공유하나 다양하게 구현해 줄 수 있음
  - 기능적으로 유사한 클래스들을 자식 클래스로 묶어줄 때 사용
- 스테레오타입《interface》를 이용하거나 동그라미 모양으로 인터페이스를 표시함
- 인터페이스와 이것을 상속받는 클래스 사이에는 구현 관계를 가짐
  - 표기는 상속과 유사하나 점선으로 표현함
  - 또는 동그라미에 실선으로 연결시킴

10. 인터페이스 (2/2)

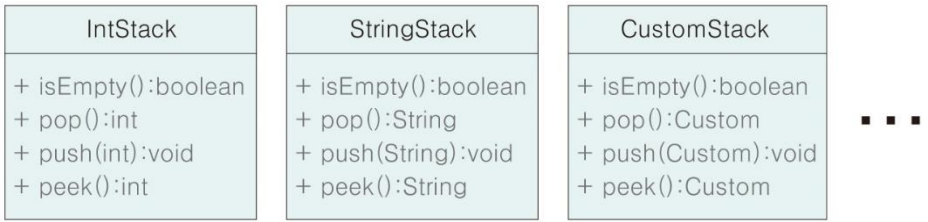
✚ 인터페이스의 사용 이유

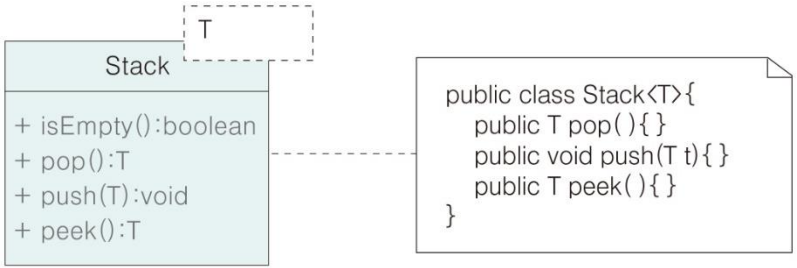
- ✕ 클래스가 제공하는 서비스와 실제 구현을 분리시킴
- ✕ 인터페이스만 변경되지 않는다면 내부 구현이 바뀌어도 인터페이스를 사용하는 클래스에 영향을 주지 않음
- ✕ 인터페이스를 사용하는 클래스는 인터페이스를 통해 인터페이스의 구현 클래스와 관계를 가져야 함



11. 템플릿 (1/2)

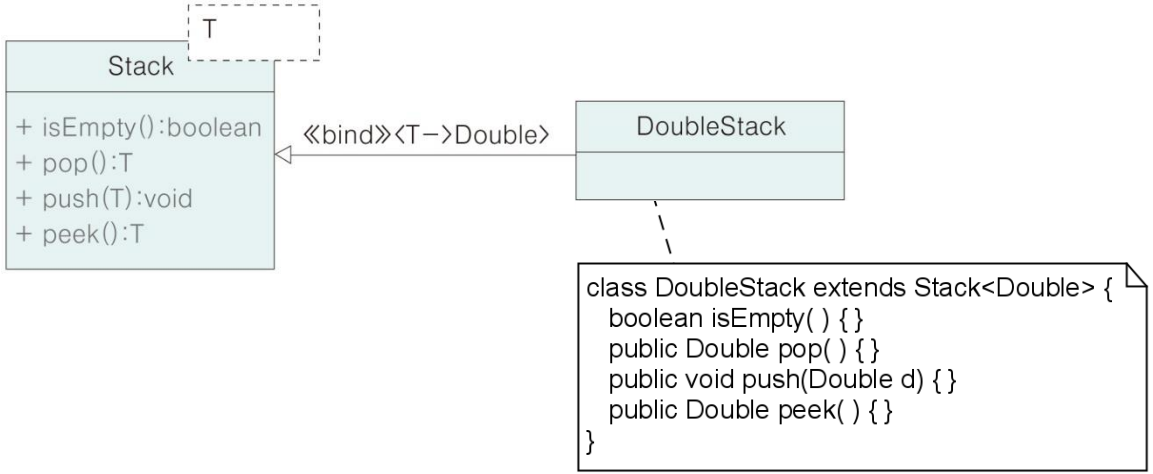
- 타입 파라미터를 가지고 있는 클래스
  - 클래스에서 다루는 데이터의 타입을 미리 정하지 않음
- 클래스의 재사용성을 높이기 위한 것으로 제너릭(generic) 이라고도 함





11. 템플릿 (2/2)

- 클래스를 정의할 때 템플릿을 사용하면 타입을 미리 정할 필요가 없음
  - 중복 코드를 제거하는 효과
- 템플릿 클래스를 이용하려면 파라미터 바인딩이 필요함
  - 서브클래스에 의한 바인딩은 템플릿의 기능을 특정 클래스에 한정시킴



Chapter. 4

객체 다이어그램

1. 객체 다이어그램

실행 중 특정 시점에서 객체들이 동작하는 상황을 표현

- 클래스 다이어그램이 표현하는 정적인 구조가 아닌 실행 중 동작 상황을 표현
- 실행 중 특정 시점에서 객체들의 관련성과 객체의 상태를 보여줌
- 객체와 데이터 값으로 구성된 클래스 다이어그램
- 클래스 다이어그램의 인스탄스
- 클래스 다이어그램과 유사한 형태이나 단순함
- 4+1뷰에서 논리 뷰에 해당

2. 객체와 링크

객체 표기법

- 사각형에 객체의 이름을 표기하고 밑줄을 그음

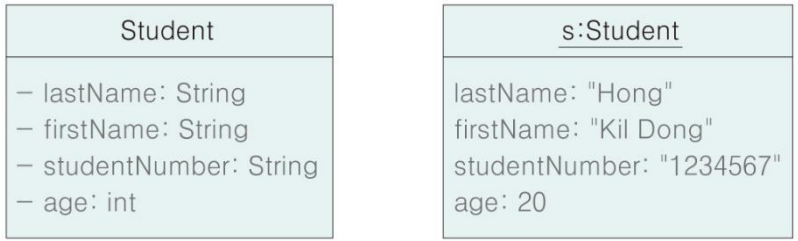
링크

- 객체 간의 연관 관계를 의미하며 실선으로 연결함
- 연결된 객체들끼리 메시지를 주고받음
- 클래스 간에 관계를 맺고 있음을 의미함
- 객체들은 해당 클래스의 제약 조건을 모두 만족해야 함

3. 상태

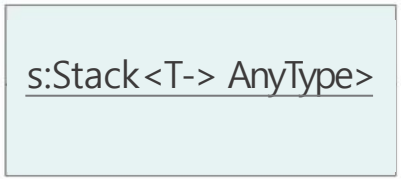
✚ 속성의 값들로 객체의 상태가 표현됨

- ✕실행 중 특정 순간에 객체가 가지고 있는 속성값들이 객체의 상태가 됨



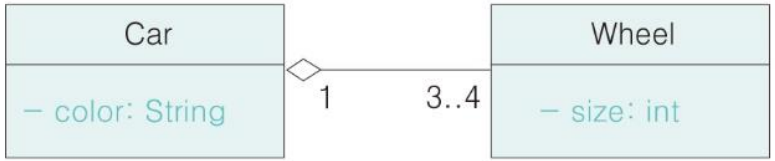
4. 템플릿과 동적 바인딩

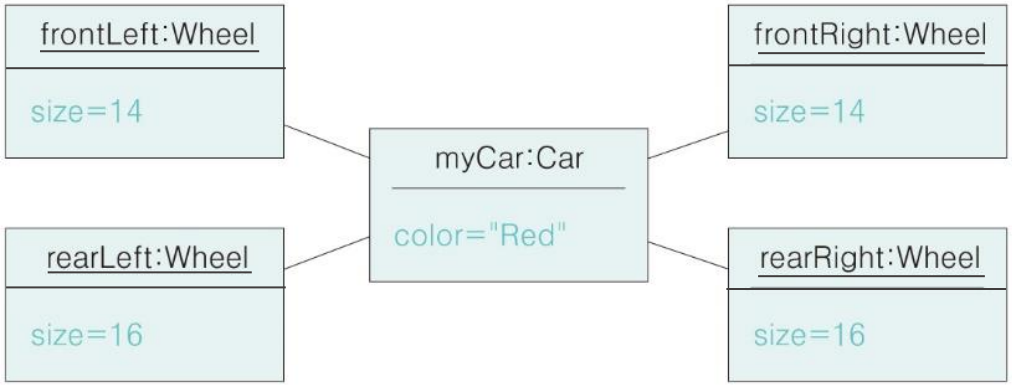
- 객체 다이어그램은 템플릿에 대한 동적 바인딩을 표현할 수 있음
- 동적 바인딩은 클래스가 아닌 객체로 만들어질 때 타입을 알려주는 것
  - ×클래스 다이어그램에서는 표현할 수 없음




```
Stack<Integer> s = new Stack<Integer>( );
s.push(3);
```

5. 클래스 다이어그램과 객체 다이어그램





다음강의

14강. 상태 머신 다이어그램