UML에 대해서

교재: 자바언어로배우는디자인패턴입문(3판)/YukiHiroshi저/영진닷컴

덕성여자대학교 컴퓨터학과
최 승 훈

01. UML에 대해서

- UML(Unified Modeling Language)
  - 시스템을 시각화하거나 요구 사항 명세서(specification)또는 설계(design)를 문서화하기 위한 표현 방법
  - 본 교재에서는, 클래스나 인스턴스의 관계를 표현하기 위해 사용함
  - 공식 웹사이트
    - www.omg.org

01. UML에 대해서

- 클래스 다이어그램
  - 클래스나 인스턴스(객체), 인터페이스 등 간의 정적인 관계를 표현한 그림

- 클래스들 사이의 주요 관계 3가지
  - 상속 관계
  - 집합(aggregation) 관계
  - 연관(association) 관계

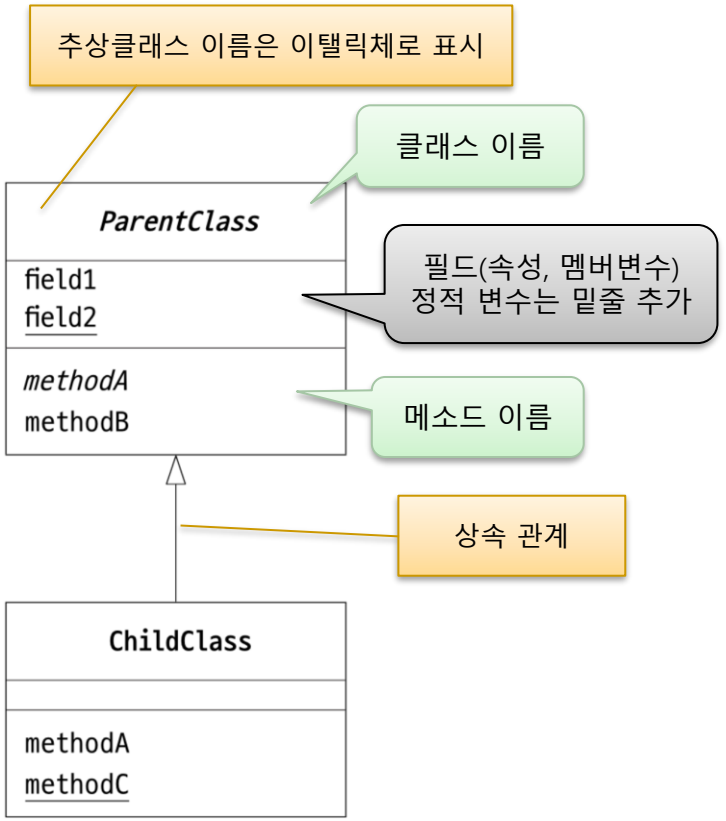
01. UML에 대해서

❑ 클래스와 계층 관계(상속관계)

그림 0-1 클래스의 계층 관계를 나타낸 클래스 다이어그램

```
abstract class ParentClass {
    int field1;
    static char field2;
    abstract void methodA();
    double methodB() {
        // …
    }
}

class ChildClass extends ParentClass {
    void methodA() {
        // ...
    }
    static void methodC() {
        // ...
    }
}
```



구현 ← 분석 또는 설계 (모델링)

01. UML에 대해서

- 앞페이지 클래스 다이어그램 예제 설명
  - ParentClass : 상위 클래스, 부모 클래스
  - ChildClass: 하위 클래스, 파생 클래스, 자식 클래스, 파생 클래스

- 부가적인 정보

  - 추상 클래스(abstract class)
    - 추상 메소드를 하나 이상 가지고 있는 클래스
    - 이탤릭체로 이름을 표현한다.
    - 이 클래스로부터 객체(인스턴스)는 생성할 수 없다.

  - 정적 필드(static field)
    - 객체가 아니라 클래스에 포함된 변수(객체마다 하나씩 생기지 않는다)
    - 밑줄을 그어 이름을 표현한다.

01. UML에 대해서

❑ 부가적인 정보(계속)

- 추상 메소드(abstract method)
  - 구현 부분이 없는 메소드 => 자식 클래스가 구현해야 한다.
  - 이탤릭체로 이름을 표현한다.

- 정적 메소드(static method)
  - 객체가 아니라 클래스에 포함된 메소드
  - 정적 메소드나 정적 필드만을 접근할 수 있다.
  - 밑줄을 그어 이름을 표현한다.

01. UML에 대해서

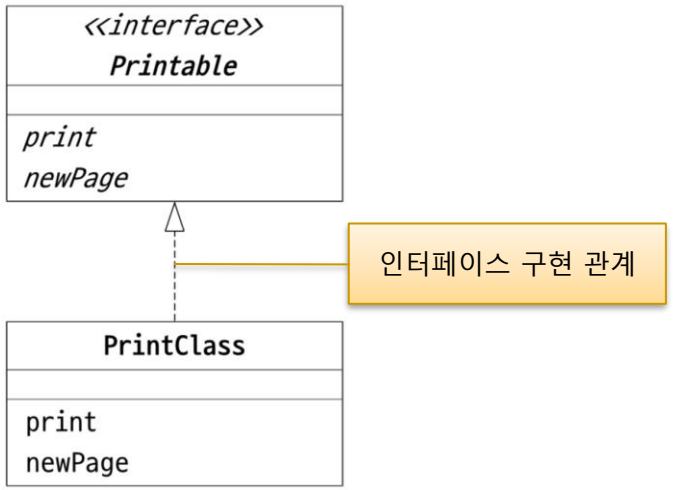
❑ 인터페이스와 구현

- 인터페이스란, 구현 부분이 생략되어 있는 메소드들의 이름만 선언되어 있는 특수한 클래스
- 설명: PrintClass 클래스가 Printable 인터페이스를 구현한다.

그림 0-2 인터페이스와 구현 클래스를 나타낸 클래스 다이어그램

```
interface Printable {
    abstract void print();
    abstract void newPage();
}

class PrintClass implements Printable {
    void print() {
        // ...
    }
    void newPage( ) {
        // ...
    }
}
```



의미: Printable 인터페이스를 구현하는 클래스는 반드시 print( )와 newPage( ) 메소드 구현을 제공한다.

01. UML에 대해서

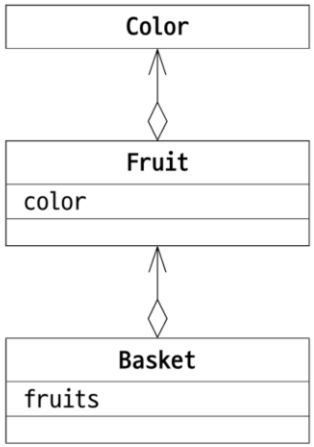
- 집합(집약, Aggregation) 관계
  - '갖고 있는 관계'는 aggregation으로 표현한다.
  - 설명:
    - Basket 클래스는 Fruit 클래스의 인스턴스를 가지고 있다. 즉, Basket 클래스에는, Fruit 클래스로 선언된 필드(변수)가 있다.
    - Fruit 클래스도 Color 클래스의 인스턴스를 가진다.

그림 0-3 집약을 나타낸 클래스 다이어그램

```
class Color {
    // ...
}

class Fruit {
    Color color;
   // ...
}

class Basket {
    Fruit[] fruits;
    // ...
}
```



01. UML에 대해서

- 액세스 제어
  - +: public 메소드나 필드(외부 클래스에서 접근 가능함)
  - -: private 메소드나 필드(외부 클래스에서 접근할 수 없음)
  - #: protected 메소드(자손 클래스 및 같은 패키지 내의 클래스 만이 접근할 수 있음)
  - ~: 같은 패키지 내의 클래스 만이 접근 가능

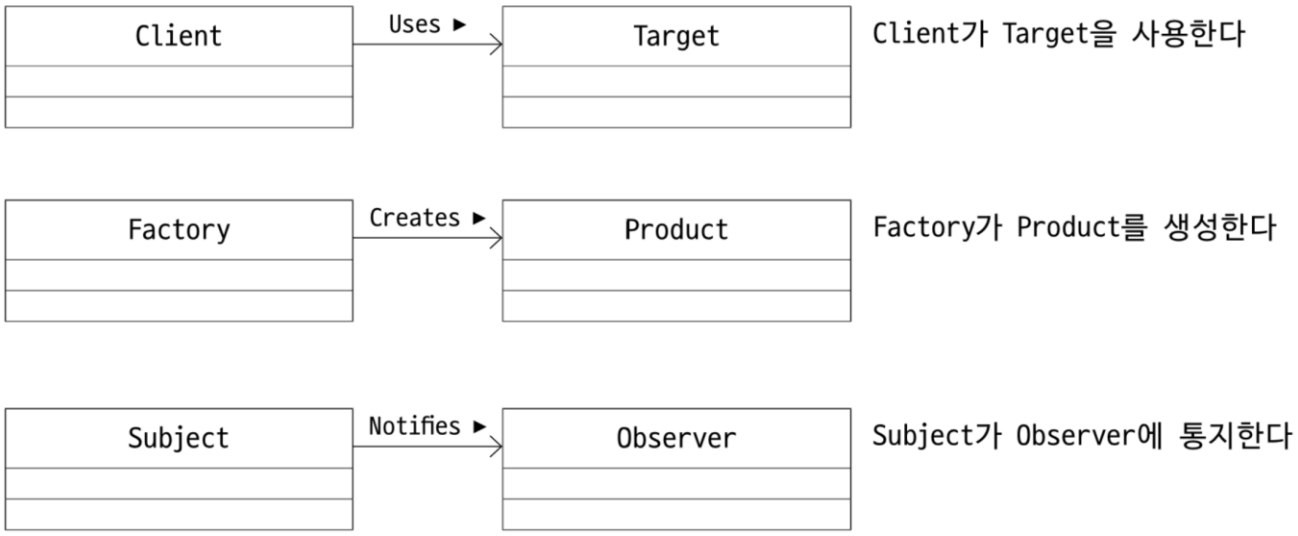
그림 0-4 액세스 제어를 명시한 클래스 다이어그램

```
class Something {
    private int privateField;
    protected int protectedField;
    public int publicField;
    int packageField;
    private void privateMethod() {
    }
    protected void protectedMethod() {
    }
    public void publicMethod() {
    }
    void packageMethod() {
    }
}
```

01. UML에 대해서

- 클래스간의 연관 관계(association)
  - 클래스간의 관계를 나타내기 위해서 클래스를 연결하고, 그 위에 관계를 나타내는 이름을 붙인다.
  - 삼각형 방향으로 해석한다.

그림 0-5 클래스의 관계



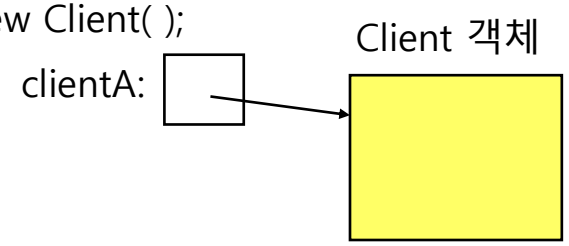
01. UML에 대해서

- 시퀀스(sequence) 다이어그램
  - 프로그램이 동작할 때, 객체들 사이의 메소드들이 어떤 순서로 실행(호출)되는지를 보여준다
    - 객체들간의 협동 과정을 보여줌(동적인 관계를 보여줌)
    - 클래스 다이어그램은 정적인 관계를 보여줌

  - :Client : Client 클래스의 익명의 객체를 의미한다.

  - clientA:Client : Client 클래스의 clientA라는 이름의 객체를 의미한다.

    - 예: Client clientA = new Client( );



clientA 변수는, Client 객체에 대한 reference(참조)를 가지고 있다.

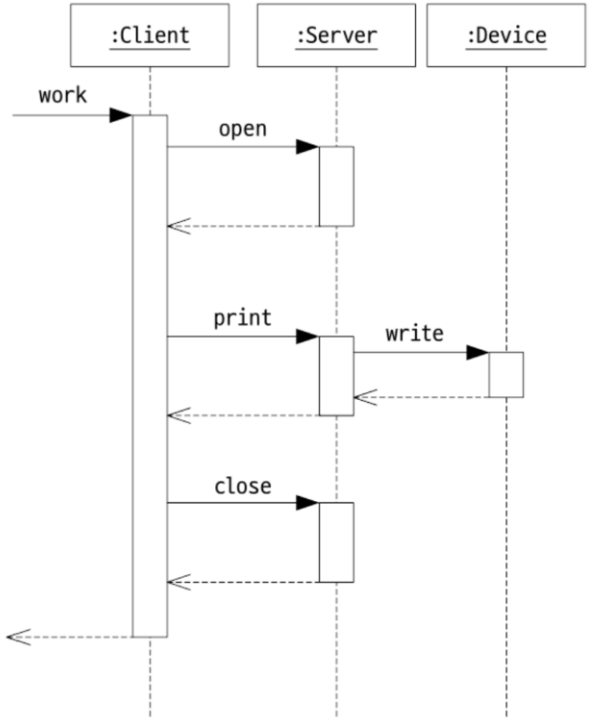
01. UML에 대해서

그림 0-6 시퀀스 다이어그램의 예 (메소드 호출)

```
class Client {
    Server server;
    void work() {
        server.open();
        server.print("Hello");
        server.close();
    }
    // ...
}

class Server {
    Device device;
    void open() {
        // ...
    }
    void print(String s) {
        device.write(s);
        // ...
    }
    void close() {
        // ...
    }
    // ...
}

class Device {
    void write(String s) {
        // ...
    }
}
```



01. UML에 대해서

- 시퀀스(sequence) 다이어그램
  - 사각형 아래의 점선은 생명선(life line)을 나타낸다
    - 인스턴스(객체)가 (메모리에) 존재하는 기간을 의미함
  - 생명선 중간의 가늘고 긴 사각형: 객체가 활동 중임을 나타낸다.
    - 즉, CPU를 얻어서 실행이 되는 상태
  - 검은색 화살표(⟶)
    - 메소드 호출
  - 점선 화살표(◂----)
    - 메소드 리턴

02. 디자인 패턴을 배우기 전에

- 디자인 패턴은 클래스 라이브러리가 아니다.
  - 클래스 라이브러리: 많이 사용되는 클래스들을 미리 만들어서 모아둔 것
    - 예: 통신 관련 클래스 라이브러리, 수학 관련 클래스 라이브러리

- 클래스 라이브러리 내에서 디자인 패턴이 사용된다.
  - 예: java.util.Calendar 클래스에의 getInstance( ) 메소드에서 Factory Method 패턴(4장)이 사용된다.

- 프로그램을 완성품으로 보지 않는다
  - 기능을 확장해 가는 것, 변경해 가는 것으로 본다.

  - 어떤 기능이 확장될 가능성이 있는가?
  - 기능을 확장할 때 어느 클래스를 수정해야 하는가?
  - 수정할 필요가 없는 것은 어느 클래스인가?

02. 디자인 패턴을 배우기 전에

- 다이어그램을 단순히 보기만 하지 말고, 그 의미를 읽어내야 한다.
  - UML은 단순한 그림이 아님
- 스스로 예제를 생각해 본다.
- 역할을 이해한다 – 백설공주의 역할은 누구인가?
  - 디자인 패턴에서 각 클래스와 인터페이스의 역할을 이해해야 함