시스템 분석 설계

UML의 이해

중간고사

- **10월 17일 11:45 ~ 12:35**

- **범위**

  - 객관식 15점 + 주관식 15점
  - 강의노트를 중심으로 배운 내용

자료흐름도(DFD)에 대한 설명으로 가장 옳지 않은 것은?

1. 구조적 분석용 문서화 도구
2. 도형 중심의 표현
3. 상향식 분할의 표현
4. 자료 흐름 중심의 표현

자료 사전에서 사용되는 기호 중 주석을 의미하는 것은?

1. { }
2. **
3. =
4. +

- **자료 사전에서 사용되는 기호 중 자료 항목이 생략될 수도 있음을 나타내는 기호는?**

1. ( )
2. #
3. &
4. !

자료 사전(Data Dictionary)에서 반복을 의미하는 기호는?

1. +
2. { }
3. [ ]
4. ( )

Contents

주요 내용

- 01 UML 용도와 특징
- 02 객체 지향 모델링

학습목표

- UML의 개념과 특징을 이해한다.
- 객체 지향 개념과 특징을 이해한다.
- 모델링 방법을 이해한다.

01 UML의 용도와 특징

UML의 탄생과 특징

- UML(Unified Modeling Language)
  - 시스템 개발을 위한 시각적인 설계 표기 제공
  - 객체 지향 시스템을 개발할 때 산출물을 명세화, 시각화, 문서화하는 데 사용
  - 개발하는 시스템을 이해하기 쉬운 형태로 표현하여 분석가, 설계자, 의뢰인이 효율적으로 의사소통할 수 있게 해줌

  ➔ UML은 표준화된 통합 모델링 언어

01 UML의 용도와 특징

UML의 탄생과 특징

- 그래디 부치Grady Booch, 제임스 럼버James Rumbaugh, 이바 야콥슨Ivar Jacobson이 UML 초안 연구
- 1997년 OMGObject Management Group, 객체 관리 그룹에서 여러 표기법을 통합하여 UML 발표

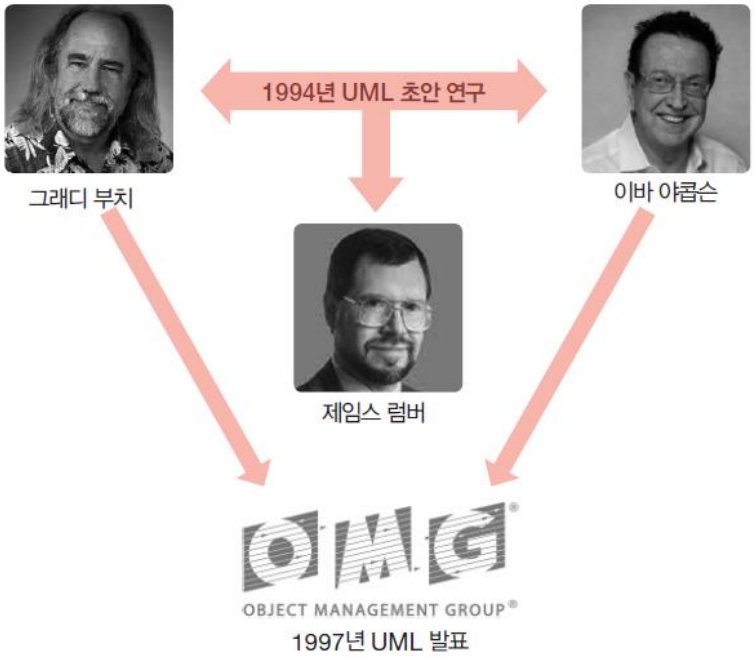


그림 1-1 UML을 개발한 사람들

01 UML의 용도와 특징

UML의 탄생과 특징

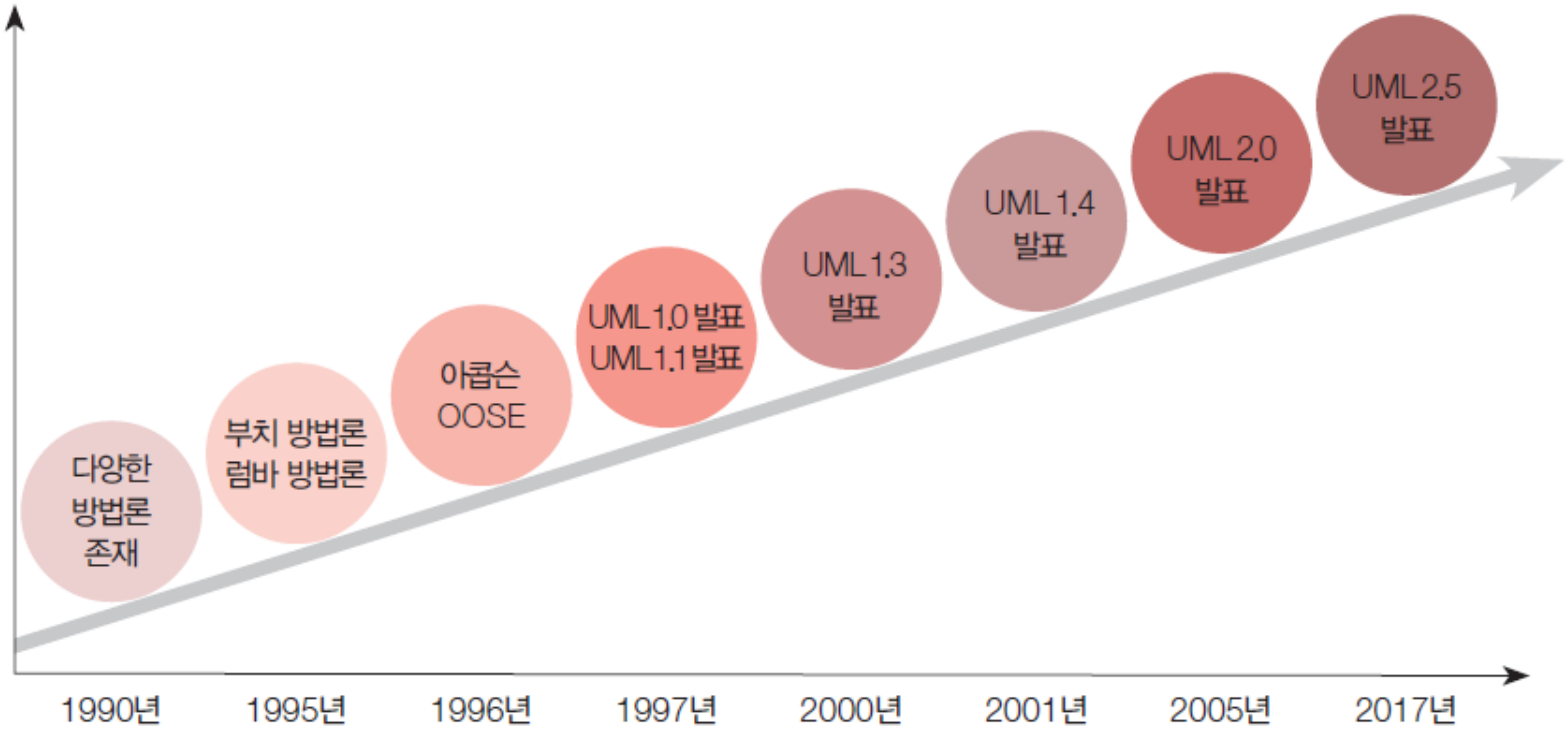


그림 1-2 UML의 발전 과정

01 UML의 용도와 특징

UML이 제공하는 표준화된 다이어그램

- 유스케이스 다이어그램Use-case Diagram
- 클래스 다이어그램Class Diagram
- 순차 다이어그램Sequence Diagram
- 통신 다이어그램Communication Diagram
- 활동 다이어그램Activity Diagram
- 상태 다이어그램State Diagram
- 컴포넌트 다이어그램Component Diagram
- 배치 다이어그램Deployment Diagram
- 패키지 다이어그램Package Diagram

01 UML의 용도와 특징

UML의 특징

- UML은 시각화Visualization 언어다.
  - 소프트웨어의 개념 모델을 시각적인 형태로 표현하며 명확히 정의된 표준화된 다이어그램을 제공
  - 이를 이용해 오류 없는 원활한 의사소통 가능
- UML은 명세화Specification 언어다.
  - 소프트웨어 개발 과정인 분석, 설계 단계의 각 과정에서 필요한 모델을 정확하고 완전하게 명세화하여 만들 수 있음
  - 명세화에서 각 다이어그램의 기호는 의미를 담고 있으며 추상적이지만 고유의 특성을 갖고 있음
- UML은 구축Construction 언어다.
  - UML은 자바Java, C++, 비주얼 베이직Visual Basic, C# 같은 다양한 프로그래밍 언어로 표현가능
  - UML로 설계된 모델을 프로그램 코드로 자동 변환할 수 있으며, 이미 구축된 소스 코드를 UML로 역변환하여 분석하는 역공학Reverse Engineering도 가능
- UML은 문서화Documentation 언어다.
  - UML은 StarUML, 투게더Together 등 케이스 툴CASE Tool을 이용하여 설계한 내용을 자동으로 문서화 가능

01 UML의 용도와 특징

```c
#include <stdio.h>
int main() {
      int a, b;
      scanf("%d %d", &a, &b);
      printf("a + b = %d", a + b);
 }
```

(a) C로 구현한 덧셈 프로그램

```java
import java.util.Scanner;
public class AddDemo {
      public static void main(String[] args) {
            Scanner in = new Scanner(System.in );
            int a = in.nextInt();
            int b = in.nextInt();
            System.out .printf("%d + %d = %d", a, b, a+b);
    }
}
```

(b) 자바로 구현한 덧셈 프로그램

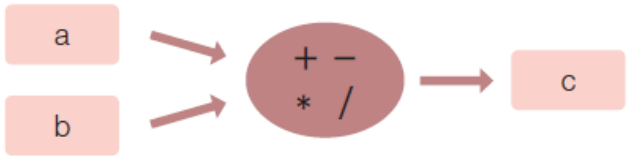
```python
def add():
      num = int(input("num1:"))
      num2 = int(input("num2:"))
      result = num + num2
      print("두수의 합은 " result)
```

(c) 파이썬으로 구현한 덧셈 프로그램

그림 1-3 다양한 언어로 구현한 덧셈 프로그램

개발하고자 하는 프로그램을 시각적으로 표현하는 것이며,
이때 의뢰자의 요구에 맞게 쉽게 수정해서
결과적으로 유지보수 기간을 줄여 생산성을 높일 수 있음

(a) 덧셈 연산 프로그램의 시각적 표현



(b) 사칙연산 프로그램의 시각적 표현

그림 1-4 프로그램의 시각적 표현

01 UML의 용도와 특징

■ 모델링이 필요한 이유

(a) 개인이 제어할 수 있는 작업 : 개집 짓기

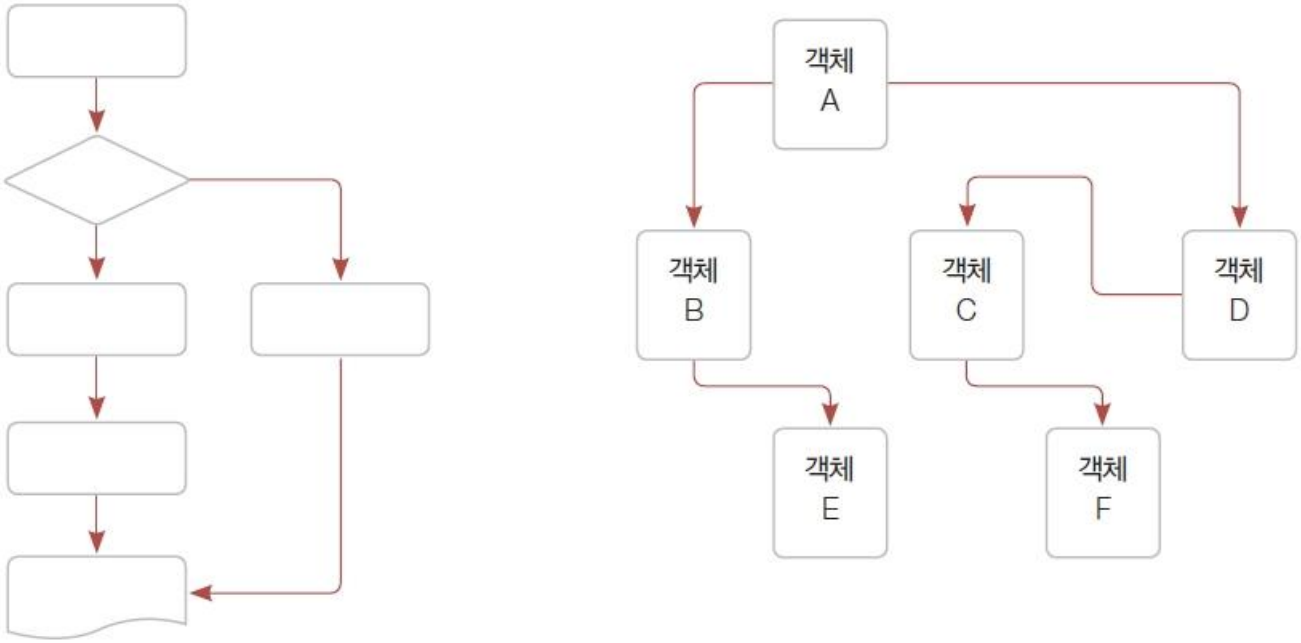
(b) 개인이 제어할 수 없는 작업 : 큰 건축물 짓기

그림 1-5 모델링이 필요한 이유

02 객체 지향 모델링

■ 객체 지향의 개념

- C++, 자바 등의 언어를 사용해 자료 구조를 중심으로 객체 를 설계한 다음 이들의 흐름을 설계하는 방식
- 객체Object,클래스Class,메시지Message를 기본요소로 함



(a) 절차 지향 방법

(b) 객체 지향 방법

그림 1-6 절차 지향 방법과 객체 지향 방법

02 객체 지향 모델링

■ 객체 지향의 개념

▪ 객체와 클래스

- 객체는 현실에 존재하는 모든 것 (구체적)
- 클래스는 개념적으로 객체를 생성 할 수 있는 틀 (개념적)
- 클래스는 그 자체만으로 사용 할 수 없음

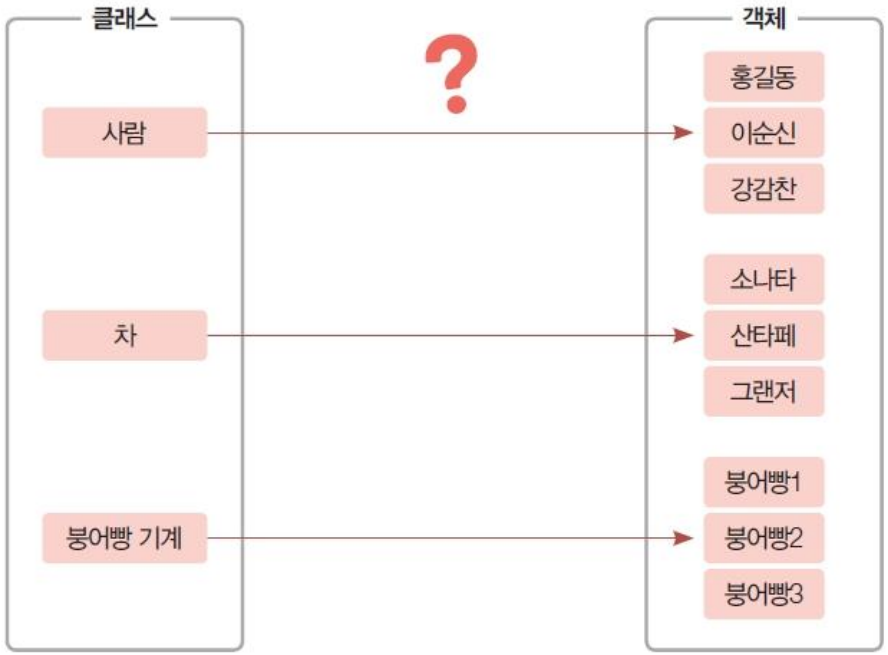


그림 1-7 클래스와 객체의 관계

그림 1-8 클래스와 객체의 예 : 붕어빵 기계와 붕어빵

02 객체 지향 모델링

■ 객체 지향의 개념

▪ 객체와 클래스

- 아래한글 실행 아이콘= 클래스, 빈 문서 1, 2, 3, 4, 5 등은 객체
- 실행 아이콘만으로는 문서 작성 불가

그림 1-9 클래스와 객체의 예 : 아래한글의 빈 문서 생성

02 객체 지향 모델링

객체 지향의 개념

객체와 클래스

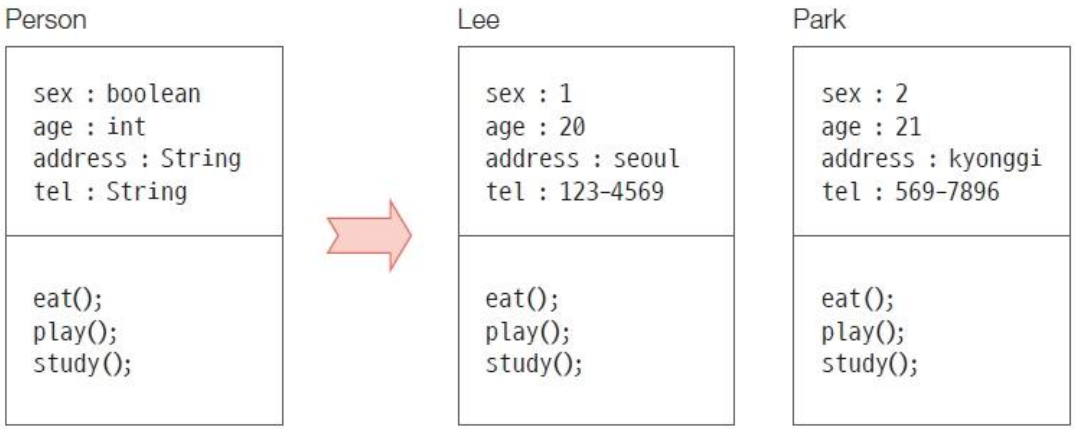


그림 1-10 프로그램에서 클래스와 객체의 예

```
Class A{
  a1();
  a2();
}
```

```
Class B{
  b1();
  b2();
}
```

```
Class C {
main() {
  A s1, s2;
  B x1, x2;

  s1.a1();
  s1.a2();
  s2.a2();
  x1.b1();
  x1.b2();
  x2.b2();
  }
}
```

객체 생성

그림 1-11 자바 프로그램에서 객체 생성

02 객체 지향 모델링

■ 객체 지향의 개념

▪ 메시지

- 객체 간의 상호작용 수단
- 한 객체가 다른 객체에 특정 작업을 요청하는 신호
- 메시지를 보내는 객체는 송신 객체Sender
- 메시지를 받아서 동작을 수행하는 객체는 수신 객체Receiver

그림 1-12 메시지 전달

02 객체 지향 모델링

■ 객체 지향의 특징

▪ 추상화

- 특정 측면을 강조하여 나타내는 것
- 객체 지향에서는 클래스를 이용하여 실세계에 대응하는 추상 모델을 만듬
- 실체화 : 추상화한 모델링을 프로그램으로 구현

그림 1-13 추상화의 개념

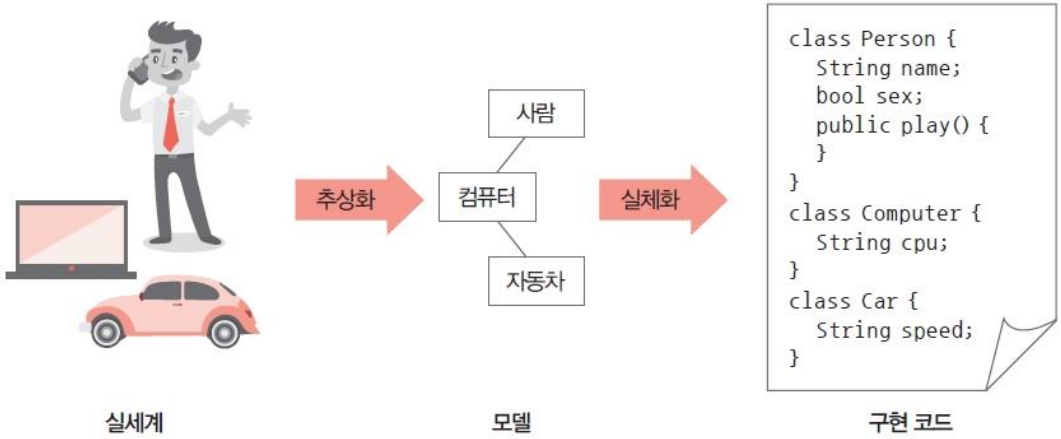
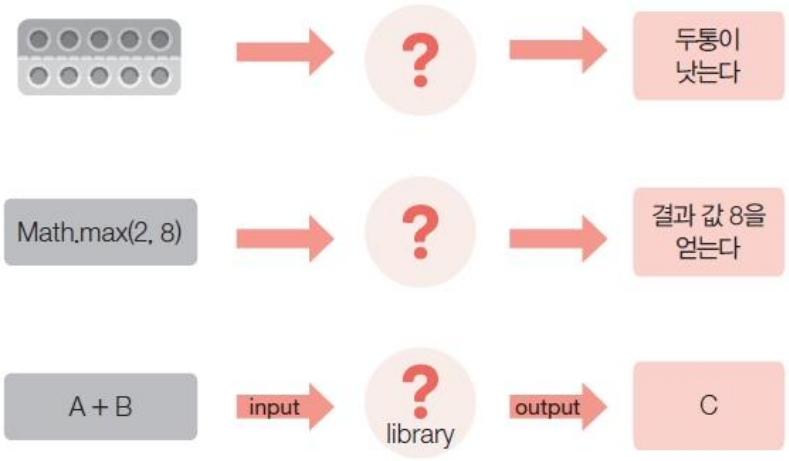


그림 1-14 추상화와 실체화의 예

02 객체 지향 모델링

■ 객체 지향의 특징

- ▪ 캡슐화
  - 데이터와 처리담당 오퍼레이션이 한 틀 안에서 결합되어 객체라는 단위로 묶여 사용되는 것
  - 캡슐화를 통해 정보 은닉Information Hiding 가능
  - 보다 높은 독립성, 유지보수성, 향상된 이식성 제공



수행 과정을 알 필요가 없고, 입력과 결과만 안다.

그림 1-15 캡슐화의 개념

02 객체 지향 모델링

■ 객체 지향의 특징

▪ 상속

- 프로그램을 쉽게 확장할 수 있도록 도와주는 수단
- 상속은 객체 지향 패러다임에서만 구현가능
- 상속은 정보를 공개하고 재사용하는 개념
  - 상세화 : 상위 클래스의 속성을 상속받아 하위 클래스에서 실체화 하는 관계
  - 일반화 : 하위 클래스의 공통 특성을 추상화 하여 상위 클래스로 정의하는 것

그림 1-16 일반화와 상세화로 표현된 상속

02 객체 지향 모델링

■ 객체 지향의 특징

▪ 상속

- 자바로 상속을 표현할 때는 [그림 1-18]과 같이 extends 키워드 사용

그림 1-17 상속을 표현한 클래스 다이어그램

```
class A {
     f1();
}
class B extends A {
     f2();
}

main() {
     B x;
     x.f2();
     x.f1();
}
```

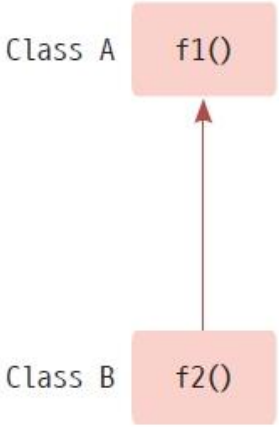


그림 1-18 자바로 구현한 상속 예

02 객체 지향 모델링

■ 객체 지향의 특징

- 다형성
  - 여러 클래스에 같은 이름의 함수가 존재하지만 동작은 다르게 수행하는 것
  - 하위 클래스에서는 그들만의 고유한 속성과 오퍼레이션 재정의 필요
  - 객체 지향 언어에서 메서드 오버라이딩Method Overriding 방식으로 구현

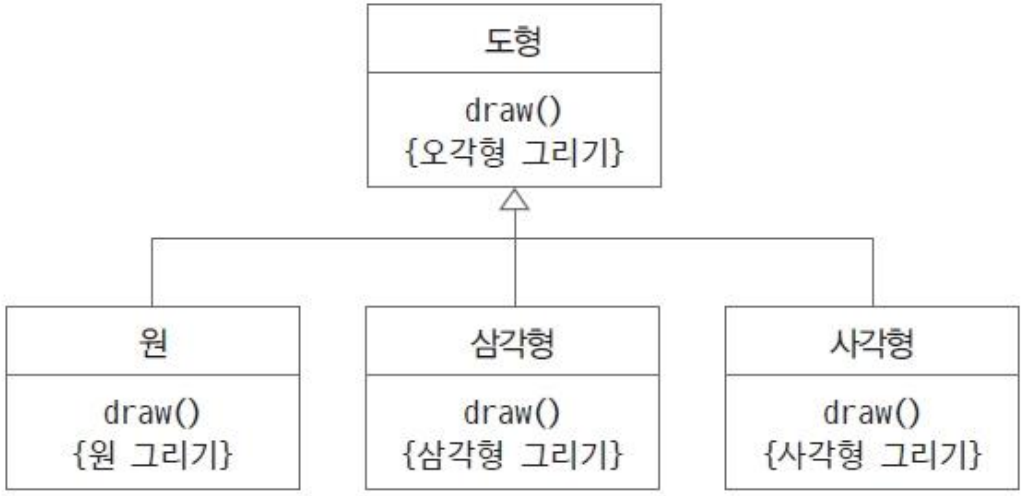


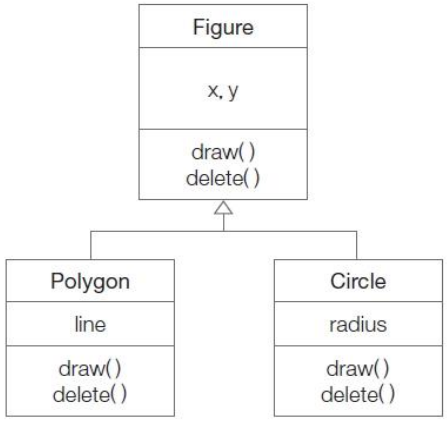
그림 1-19 다형성의 예

02 객체 지향 모델링

■ 추상 클래스와 인터페이스

▪ 추상 클래스

- 추상 클래스에는 클래스의 명칭과 메서드는 있으나 메서드의 처리 내용은 없음
- 따라서 상속을 통해서 메서드가 구현Implementation
- 추상 클래스는 추상 메서드 외에 일반적인 속성과 메서드를 가질 수 있음
- 메서드의 다형성을 지원




```
abstract class Flgure {
     public void setCoo(point x, point y) {
          my_X = x;
          my_Y = y;
     }
     public abstract void draw();
     public abstract void delete();
}

class Polygon extends Figure {
     int line;
     public void draw() {.....}
     public void delete() {.....}
}

class Circle extends Figure {
     int radius;
     public void draw() {.....}
     public void delete() {.....}
}
```

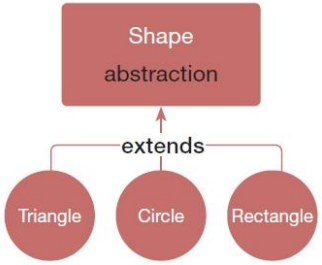
그림 1-20 추상 클래스의 예

02 객체 지향 모델링

■ 추상 클래스와 인터페이스

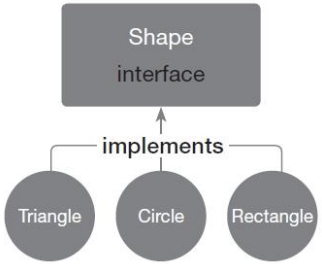
▪ 인터페이스

- 상수와 추상 메서드만 가짐
- 여러 개의 인터페이스로부터 상속받을 수 있기 때문에 다중 상속의 기능을 제공
- 추상 클래스와 달리 속성을 가질 수 없으며, 메서드의 구현도 정의할 수 없음



(a) 추상 클래스

```
abstract class A {
  int k;
  abstract void draw();
}
class B extends A {
  draw {삼각형 그리기};
}
class C extends A {
  draw {원 그리기};
}
```



(b) 인터페이스

```
interface A {
  int k=3;
  void draw();
}
class B implements A {
  draw() {
    사각형 그리기;
  }
}
```

그림 1-21 추상 클래스와 인터페이스

02 객체 지향 모델링

■ 모델링 개념

▪ 모델링

- 시스템을 구축할 때 개발자가 고민하고 결정하는 모든 활동
- 구현 단계 이전의 요구 사항 정의, 분석, 설계에서 수행하는 활동
- 모델 : 모델링의 결과물
- CASE 툴: 모델링을 전문적으로 지원하는 툴
  - Ex) StarUML, 로즈Rose, 투게더Together 등

표 1-1 모델링과 프로그래밍

구분	모델링	프로그래밍
목적	구축할 시스템의 모습 정의	시스템의 실제 구현
세부 수행 활동	요구 사항 정의, 분석, 설계	소스 코드 편집, 컴파일, 시험, 디버깅
결과물	모델	소스 코드를 포함한 구현된 시스템
표기법	모델링 언어(UML, ERD, DFD)	프로그래밍 언어(자바 , C++)
자원 툴	CASE 툴(StarUML, 로즈Rose, 투게더Together)	개발 툴(Jbuilder, Visual Studio, .Net)

02 객체 지향 모델링

모델링 방법

- 부처 방법론Booch Method
  - 설계 중심의 방법론으로 시스템을 몇 개의 뷰View로 분석할 수 있다고 보고 뷰를 모델 다이어그램으로 표현
  - 거시적 개발 프로세스와 미시적 개발 프로세스를 모두 포함하고 단계적 접근과 자동화 툴 지원
  - 객체 지향 방법론에 대한 광범위한 이론적 배경을 제시하여 더 넓게 바라볼 수 있는 안목을 제공

- 야콥슨의 OOSE EObject-Oriented Software Engineering
  - 유스케이스를 강조한 방법론
  - 유스케이스는 외부 행위자와 상호작용하는 시스템의 요구 사항을 정의하고, 이렇게 정의된 유스케이스는 개발, 테스트, 검증 단계에서 사용됨
  - 방법론이 복잡하여 초보자에게는 어렵지만, 큰 규모의 시스템을 개발하는 데 효율적

02 객체 지향 모델링

모델링 방법

- 럼바의 OMT Object-Modeling Technique
  - 하나의 시스템을 기술하기 위하여 객체 모델Object Model, 동적 모델Dynamic Model, 기능 모델Functional Model의 세 가지 모델을 사용
  - 시스템 분석에서 이 세 가지 모델을 이용하여 시스템이 요구하는 객체를 기술
  - 객체 모델은 시스템에서 필요한 모델을 찾아내고 객체의 속성과 객체 간의 관계를 규명
  - 동적 모델은 객체 모델에서 나타난 객체들의 행위와 상태를 포함하는 생명주기를 나타냄
  - 기능 모델은 각 객체의 변화로 인해 다른 상태로 전이가 되었을 때 수행되는 동작들을 기술.

- UML
  - 부치 방법론과 OOSE, OMT를 하나로 합한 방법론
  - 분산 객체Distributed Objects의 표준이 되었고, OMG에서 CORBA Common Object Request Broker Architecture의 표준 분석 설계 방법론으로 채택