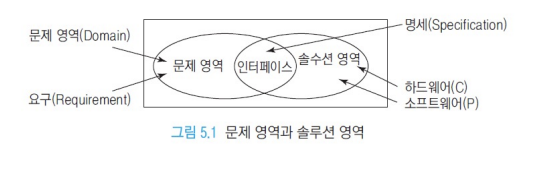
**<Chapter 5. 요구 모델링>**

**요구 모델링**

• 고객과 개발자가 무엇이 개발되고 있는지에 동의하는 것을 주된 목적으로 하는 요구 명세를 생성

• 시스템에 대한 형식적 또는 준형식적 설명을 제공

****

**모델링 기초**

• 복잡한 시스템을 다루는 방법 – 전체를 다루기에는 너무 복잡한 대상을 추상화 또는 단순화

• 모델링을 하는 이유

(1) 복잡함을 잘 관리하기 위하여

(2) 형체가 없는 소프트웨어의 구조를 시각화 하기 위하여

(3) 다른 사람과 커뮤니케이션 하기 위하여

(4) 문제 도메인 및 제품 요구 사항을 이해하기 위하여

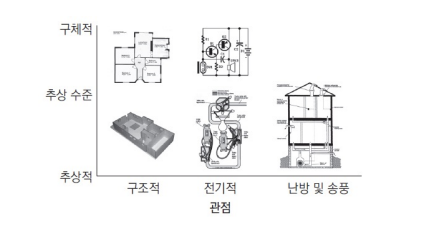
(5) 개발 중인 시스템을 이해하기 위하여

(6) 구현하기 전에 잠재적 솔루션을 실험해보기 위하여

(7) 기존 시스템의 문서화

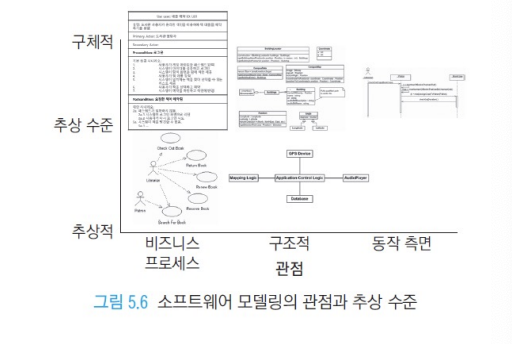
**관점과 추상화 수준**

• 모델은 특정 관점(perspective)과 추상화 수준(abstraction level)에 따라 달라짐

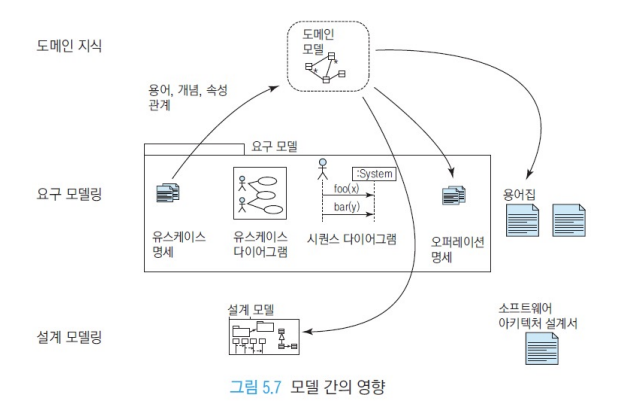


**소프트웨어와 모델링**

• 그래픽 기호와 주석으로 구성된 시각적 다이어그램



**모델 사이의 관계**



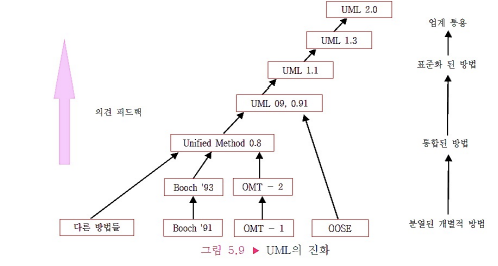
**UML(Unified Modeling Language)**

• 객체지향 소프트웨어를 모델링 하는 표준 그래픽 언어 – 시스템의 여러 측면을 그림으로 모델링 – 하드웨어의 회로도 같은 의미

• UML은 소프트웨어 모델링의 공통 언어

**UML의 역사**

• UML은 OMT(Object Modeling Technique)[Rumbaugh, 1991]와 Booch[Booch,1994], OOSE(Object-Oriented Software Engineering)[Jackson, 1992] 방법의 통합으로 만들어진 표현



**UML 다이어그램**

• 시스템의 모델링은 구조적 관점, 동작 관점으로 구성 – UML 주요 다이어그램(pp.189)

텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**UML 모델링 과정**

요구를 사용 사례로 정리하고 사용 사례 다이어그램을 작성

➋ 클래스 후보를 찾아내고 개념적인 객체 모형을 작성

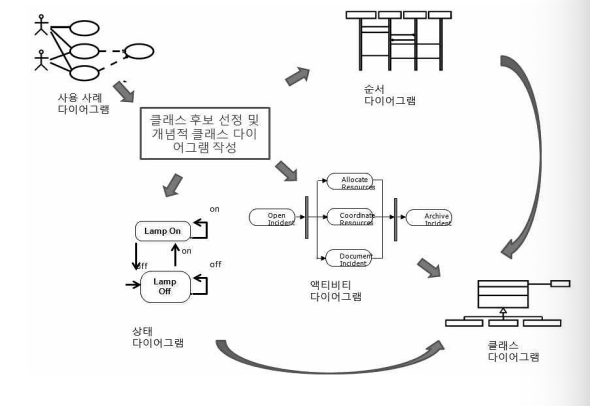
➌ 사용 사례를 기초하여 순서 다이어그램을 작성

➍ 클래스의 속성, 오퍼레이션 및 클래스 사이의 관계를 찾아 객체 모형을 완성

➎ 상태 다이어그램이나 액티비티 다이어그램 등 다른 다이어그램을 추가하여 UML 모델을 완성

➏ 서브시스템을 파악하고 전체 시스템 구조를 설계

➐ 적당한 객체를 찾아내거나 커스텀화 또는 객체를 새로 설계



**정적 모델링**

• 정적 모델 – 객체들의 공통 구조와 동작들을 추상화 시킨 것

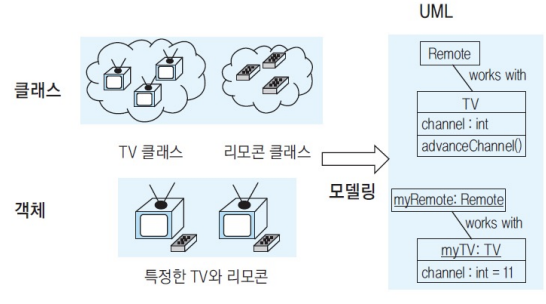
• 객체지향 기본 개념의 이해가 필요 – 객체와 속성, 연관, 집합, 상속, 다형성

• 클래스 다이어그램이 대표적 – 클래스 및 클래스 사이의 관계를 표현 – 도메인 개념과 속성

**객체와 클래스**

• 객체: 상태, 동작, 고유 식별자를 가진 모든 실체

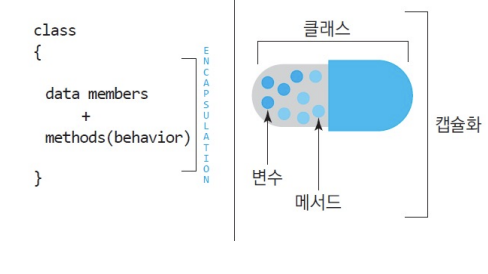
• 클래스: 공통 속성을 공유하는 객체 집합에 대한 정의



**캡슐화**

• 객체의 속성 부분과 오퍼레이션 부분을 하나로 모아서 단위화

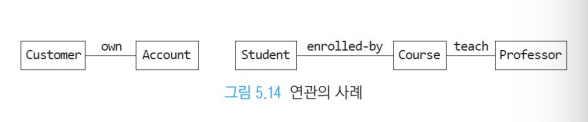
• 정보은닉



**연관**

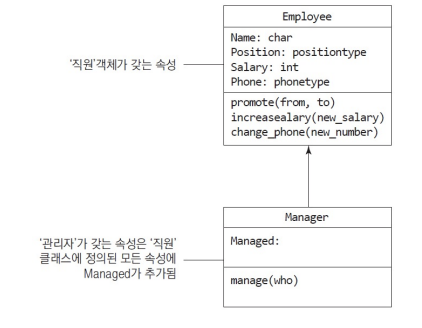
• 연관 – 서비스를 제공하는 객체와 서비스를 요청하는 객체가 상호작용하는 관계(pp.196)

• 가시성 – 객체의 접근 가능성 – 연관이 있는 객체는 서로 알고 접근 방법이 필요



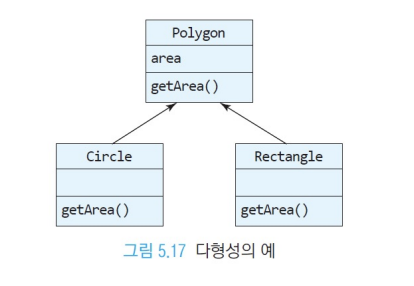
**상속**

• 일반화된 클래스가 갖는 속성과 연산을 하위 개념의 구체화된 클래스가 그대로 물려받는 것



**다형성**

• 같은 이름의 메시지를 다른 객체 또는 서브 클래스에 호출할 수 있는 특징 – twoDSHape.getArea():



**클래스의 표현**

• 클래스 심볼

– 세 개의 부분으로 나누고

– 클래스의 이름, 중간에는 클래스의 속성, 아래 부분은 오퍼레이션을 적음

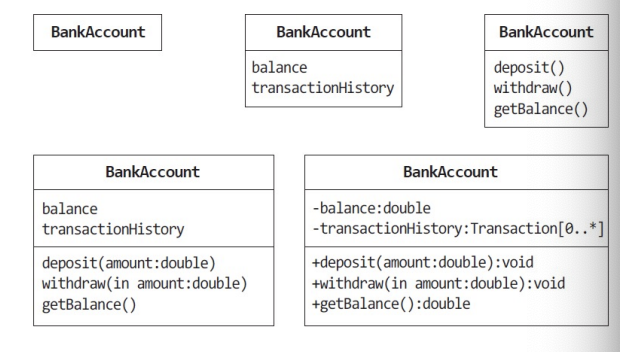
– 추상클래스는 이탤릭체, 인터페이스 클래스는 <<interface>>추가

• 속성 : 객체가 가지는 모든 필드를 포함

• 오퍼레이션 / 메소드 – 아주 흔한 메소드(get/set)는 생략

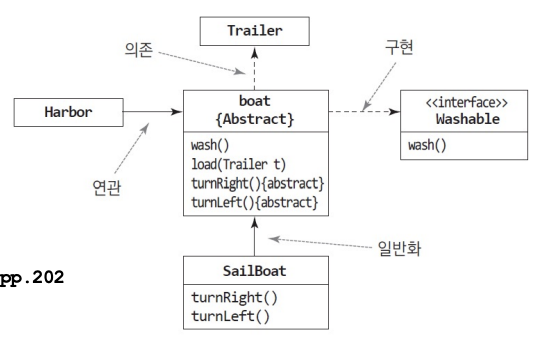
**표현 수준**

• 모델링이 진행되면서 상세화



**관계의 표현 p.202**

(1) 연관 (2) 상속 (3) 의존 (4) 구현



**(1) 연관**

테이블이(가) 표시된 사진

자동 생성된 설명

**(2) 상속**

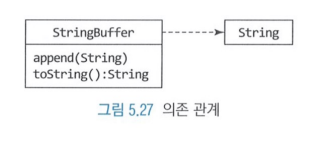
– 일반화(generalization) 관계로 모델링

– 대체 가능성

**(3) 의존**

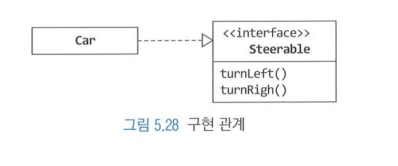
- 결합력이 가장 약하다

– 한 클래스 변경이 다른 클래스의 변경으로 연결 가능



**(4) 구현**

– 클래스가 인터페이스(메소드 정의, 구현 누락)를 구현



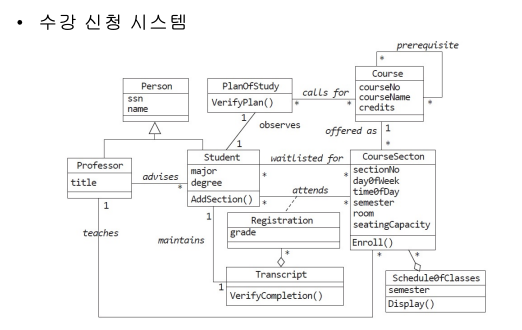
**클래스 다이어그램 표현 요소**

테이블이(가) 표시된 사진

자동 생성된 설명

**사례**

-수강 신청 시스템



**동적 모델링**

• 동적 측면 – 소프트웨어가 실행될 때 변경 될 수 있는 뷰 – 시간의 함수로만 이해 – 예: 객체 간 상호작용 패턴

• 정적 다이어그램을 보완

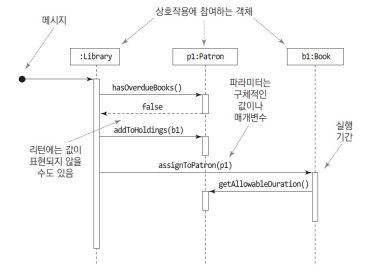
• 상호작용 다이어그램

1. 시퀀스 다이어그램

2. 협동 다이어그램

**(1) 시퀀스 다이어그램**

• 시스템의 동작을 정형화하고 객체들의 메시지 교환을 울타리 형태로 시각화하여 나타낸 것



**요소**

테이블이(가) 표시된 사진

자동 생성된 설명

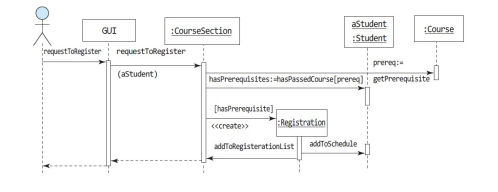
**시퀀스 다이어그램 작성 과정**

• Step 1. 참여하는 객체를 파악

• Step 2. 파악한 객체를 X축에 나열하고 라이프라인을 그음

• Step 3. 사용사례에 기술된 이벤트 순서에 따라 객체의 메시지 호출을 화살표로 나타냄

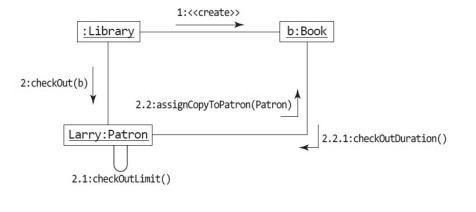
• 사례: 수강신청 유스케이스



**협동 다이어그램**

• 두 가지 조합

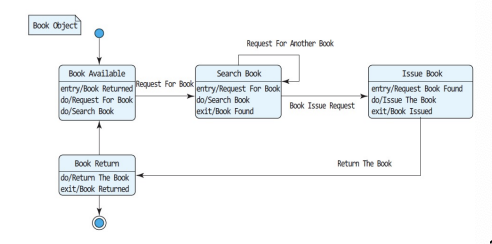
– 상호작용에 필요한 객체들 간의 링크를 포함한 객체 다이어그램 –상호작용을 정의하는 객체 간의 메시지(pp. 209)



**상태 다이어그램**

• 동작을 수신 이벤트와 이에 대한 응답을 기반으로 상태 사이의 전환으로 모델링

• 예: 도서관 시스템에서 책의 상태



**제어 모델링**

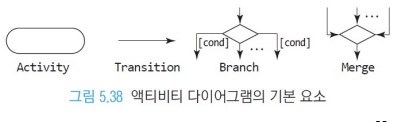
• **액티비티 다이어그램** – 액티비티 사이의 제어흐름을 보여 주는 일종의 흐름도

• **액티비티** – 계산 또는 프로세스

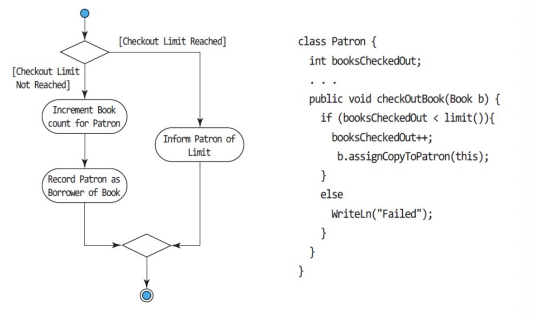
• **전환** – 액티비티에서 다른 액티비티로 제어가 넘어감

• **분기** – 진위 조건

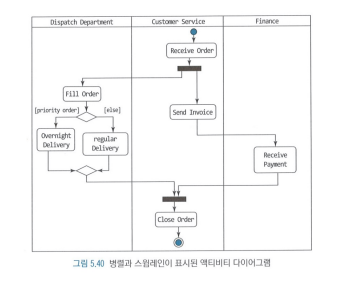
(액티비티 다이어그램의 기본 요소)



**액티비티 다이어그램**



• 병렬과 스윔레인이 표시된 엑티비티 다이어그램



**모델 검증**

• 모델 검증 방법

– 리뷰: 워크스루, 인스펙션

– 테스팅 – 정형적 방법

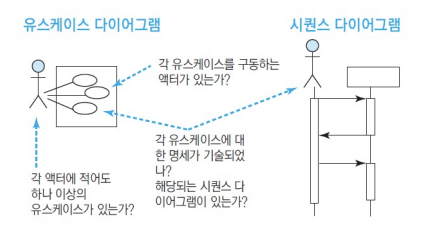
– 프로토타이핑

– 요구 추적

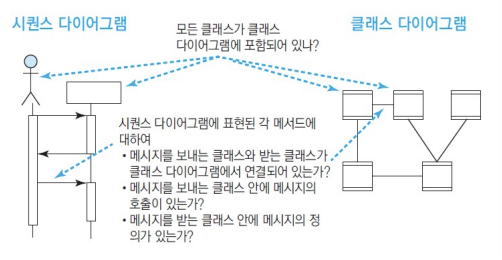
**일관성 체크**

• 유스케이스 다이어그램과 시퀀스 다이어그램

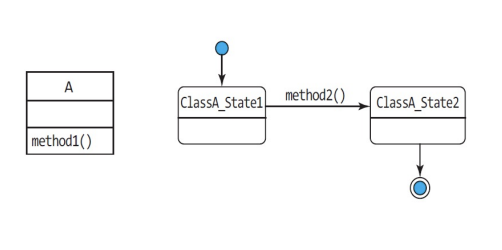
– 유스케이스에 대한 명세가 기술되어 있고 매칭되는 시퀀스 다이어그램이 있는지 체크



- 시퀀스 다이어그램 안에 포함된 클래스와 메시지가 클래스 다이어그램에 빠지지 않고 표현되었는지 체크



-상태 다이어그램과 클래스 다이어그램을 크로스체크



-리뷰 체크리스트

텍스트이(가) 표시된 사진

자동 생성된 설명