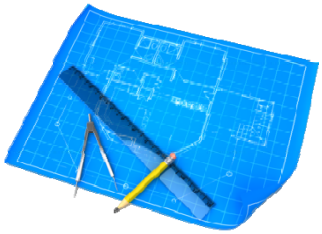


객체지향 개발 절차와 UML

NOTE 02

한국기술교육대학교 컴퓨터공학부 김상진

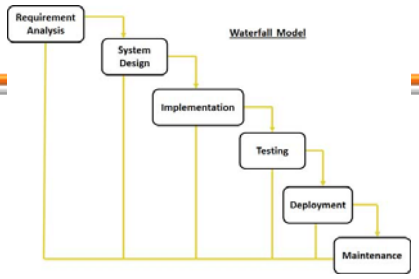
sangjin@koreatech.ac.kr
www.facebook.com/sangjin.kim.koreatech

교육목표

- 객체지향으로 소프트웨어를 개발하는 절차를 이해함
  - 반복적 방법론: 에자일 개발 방법론
- UML(Unified Modeling Language) 표기법과 UP(Unified Process) 중 수업 중 관련 있는 내용 이해
  - Use case diagram
  - Use case specification
  - 클래스 관계도
  - 시퀀스 다이어그램

A critical ability in OO development is to skillfully assign responsibility to software objects

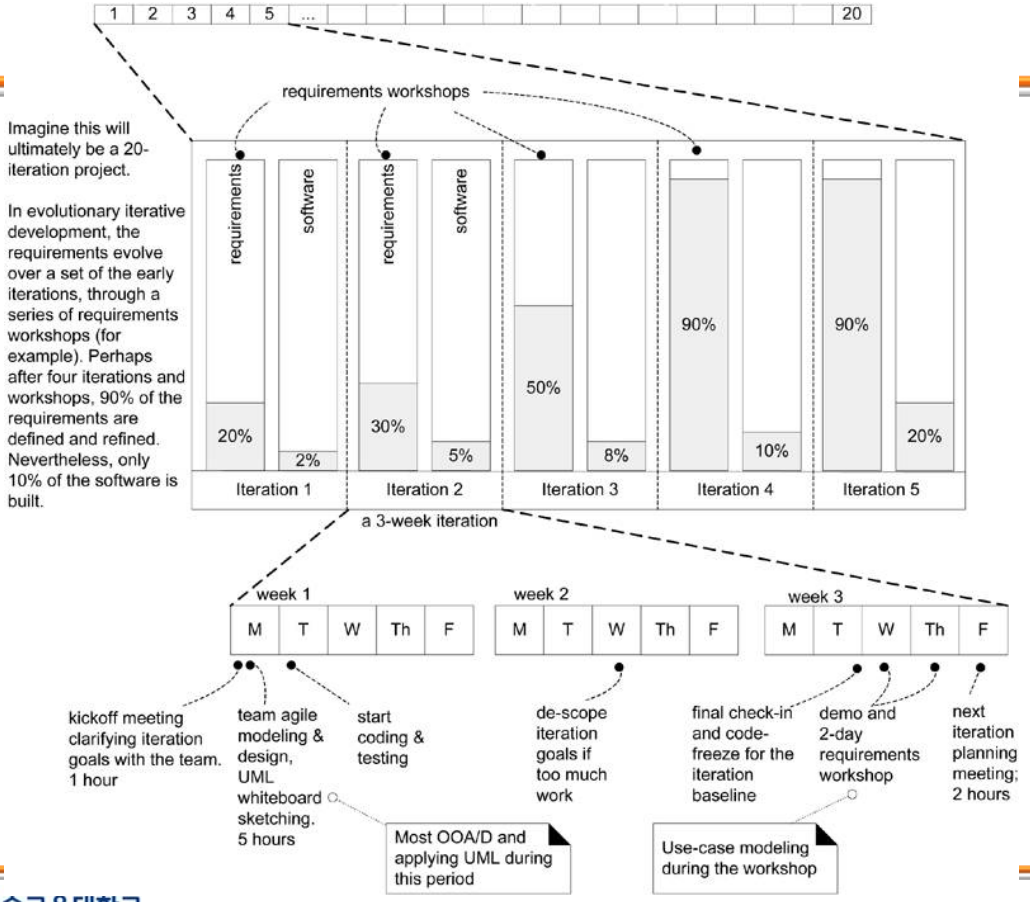
소프트웨어 개발



- 소프트웨어 개발 과정 (폭포수 모델)
  - 문제 정의
  - 요구사항 분석
    - 사용자 요구사항 분석,
      기능/비기능 요구사항 분석, 사용 시나리오 분석
    - 사용자 인터페이스 분석, 데이터 분석
  - 설계
    - OOD: 클래스 도출, 클래스간의 관계 설정, 시퀀스 다이어그램
  - 구현
  - 통합
  - 시험/평가: 구현 전에 방법 결정 또는 테스트 프로그램 개발
  - 배포: 최근 오픈 마켓 때문에 학생 때 배포/운영이 용이함
  - 유지보수

반복적 개발 방법론

- 반복적(iterative) 개발 방법론
  - 반복(iteration)이라고 하는 일련의 짧은 고정된 길이의 작은 프로젝트 단위로 전체 개발을 나누어 진행하며, 각 반복은 독립적으로 테스트하며, 통합되어 실행될 수 있는 전체 시스템의 부분 요소임
  - 각 반복은 별도의 요구사항 분석, 설계, 구현, 테스트 활동을 포함함
    - 반복의 기간은 보통 2주~6주임
  - 시스템은 연속적으로 확대되고 개선되므로 이와 같은 개발 방법론을 점증(incremental) 개발론 또는 진화적(evolutionary) 개발론이라 함
  - 이와 같은 개발의 장점은 변화에 능동적으로 대응하기 쉽다는 것임
    - 조기 피드백을 통한 개선은 고객과 개발자에게 모두 이득
  - 위험 기반(risk driven), 고객 기반 위주로 반복을 선택
    - 가장 위험이 높은, 고객이 가장 원하는 기능부터 구현
      - 핵심 요소부터 구현



에자일 개발 방법론 (1/3)

에자일 소프트웨어 개발 선언

우리는 소프트웨어를 개발하고, 또 다른 사람의 개발을
도와주면서 소프트웨어 개발의 더 나은 방법들을 찾아가고
있다. 이 작업을 통해 우리는 다음을 가치 있게 여기게 되었다:

공정과 도구보다 **개인과 상호작용**을
포괄적인 문서보다 **작동하는 소프트웨어**를
계약 협상보다 **고객과의 협력**을
계획을 따르기보다 **변화에 대응하기**를

가치 있게 여긴다. 이 말은, 왼쪽에 있는 것들도 가치가 있지만,
우리는 오른쪽에 있는 것들에 더 높은 가치를 둔다는 것이다.

에자일 개발 방법론 (2/3)

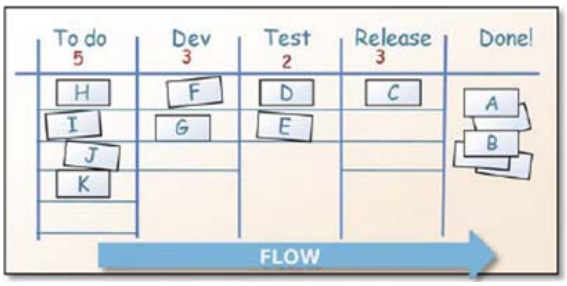
- 아무런 계획이 없는 방법과 계획이 지나치게 많은 방법 사이에 타협점을 찾고자 하는 방법론
- 기존 전통 방법이 계획에 너무 의존적이고 형식적인 절차에 따르는데 필요한 시간과 비용이 많이 소요되는 문제점을 극복하기 위해 개발된 방법론
- 문서 중심에서 코드 중심으로
- 일정한 주기를 가지고 끊임없이 프로토타입을 만들어내며 그때 그때 필요한 요구를 더하여 수정하여 큰 규모의 소프트웨어를 개발해 나가는 적응적 개발방법론임
  - 리펙토링의 반복적으로 적용하기 쉬운 개발방법론
- 익스트림 프로그래밍(eXtreme Programming, XP), 스크럼(scrum) 등 다양한 종류의 방법론이 있음

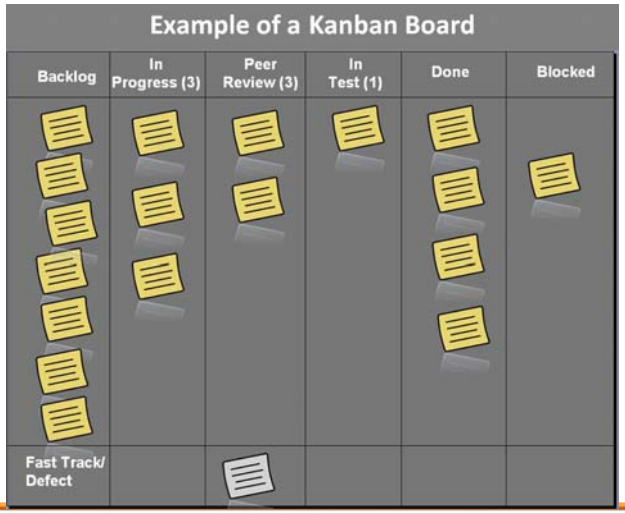
에자일 개발 방법론 (3/3)

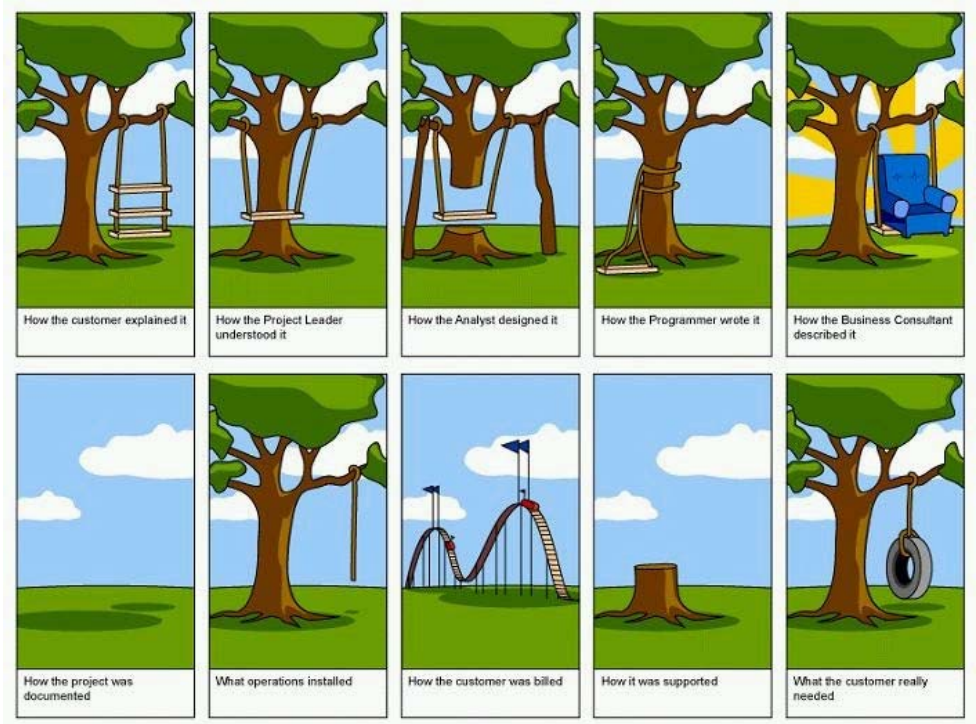
- 폭포수 모델은 순차적 진행
  - 한 단계 완료 후 다음 단계 진입
  - 실제 모든 단계를 완벽하게 완료하는 것은 불가능
  - 요구사항이 진행 과정에서 변경될 수 있다는 것을 고려하지 못함
- 에자일 개발
  - 점진적 접근 (스크럼의 경우 스프린트(Sprint) 단위)
  - 각 작은 단위에 대한 개별 검증
  - 하지만 에자일의 경우에도 설계를 전혀 하지 않는 것은 아님

Kanban Board

- 프로젝트 진행 현황을 관리하기 위한 보드
- Backlog: 모든 할 일
- To Do: 진행할 업무
- In Progress: 작업이 시작된 업무
- Done: 완료된 업무







분석

- 분석은 해결책을 찾는 것이 아니라 문제와 요구사항을 조사하는 것임
  - 해당 소프트웨어는 어떻게 사용하는 것인지, 어떤 기능이 제공되어야 하는 것인지를 고민하는 과정
  - 객체지향 분석에서는 최종 목표는 사용할 객체를 찾는 것임
- 분석 과정
  - 문제 정의
  - 요구사항 분석
  - 사용 시나리오 분석
  - 사용자 인터페이스 분석
  - 데이터 분석

검증가능 요구사항

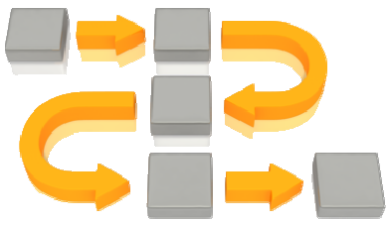
- 요구사항은 향후 검증이 가능(verifiable)해야 함
  - 충족여부를 효율적으로 확인 및 평가할 수 있어야 함
  - 나쁜 예1) 시스템은 사용하기 쉬워야 함
    - 사용하기 쉬운 여부는 어떻게 측정하나?
  - 이 측면 때문에 앞서 언급한 구체적 정량화가 필요함
  - 나쁜 예2) 고정전원 공급이 중단되어도 백업 배터리를 통해 ~~사용할 수 있어야 함~~ → 최소 20분 동안은 정상적으로 사용할 수 있어야 함

요구사항 속성

- 우선순위: 각 요구사항의 우선순위는 다음과 같이 분류 가능
  - Must have: 의무 기능 (mandatory)
  - Should have: 필요 사항 (desirable)
  - Could have: 선택 사항 (optional)
  - Want to have: 차기 버전에 고려할 사항 (future enhancement)
- 다른 속성
  - Status: Proposed, Approved, Rejected, Incorporated
  - Benefit: Critical, Important, Useful
  - Effort: 기능 구현에 소요되는 비용
  - Risk: 기능 추가에 따른 위협 (high, medium, low)
  - Stability: 변경 가능성 (high, medium, low)
  - Target Release: 기능 포함된 제품 버전

사용 시나리오 분석

- 사용 시나리오의 구성요소
  - 시나리오가 시작될 때 사용자 또는 시스템이 기대하는 것
  - 시나리오를 시작하는 외부개체
  - 일반적인 동작 흐름
    - 동작 흐름에서 발생할 수 있는 문제점과 대처방안
  - 대안 흐름
  - 동시에 수행될 수 있는 다른 행위들
  - 시나리오가 끝났을 때 시스템 상태



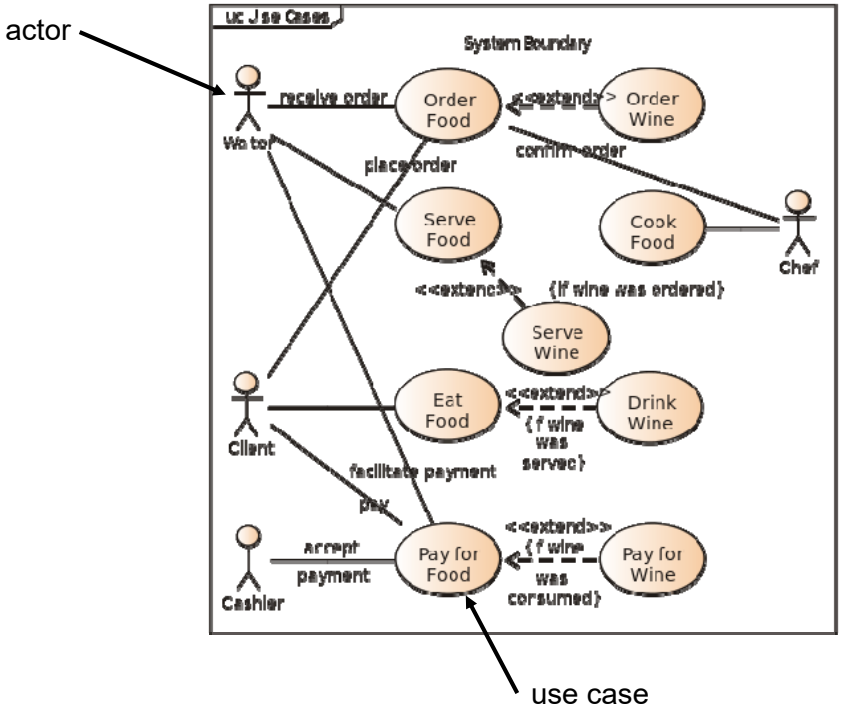
Use case 모델

- Use case 모델. 사용 시나리오 기반의 요구사항을 보충하여 주는 도구
  - Actor: 시스템을 사용하는 사람이나 장치의 역할
  - Scenario: 일련의 Actor와 시스템 간에 상호작용 또는 actor와 시스템의 행동
  - Use case: Actor들이 시스템을 이용하여 할 수 있는 것
    - 관련 있는 성공 또는 실패 시나리오들의 집합
- Use case 작성할 때에는 how에 대해 걱정하지 말고, what에 초점을 두어 작성함
- 하나의 use case는 단일 목적이 있어야 함
  - 목적이 다르면 별도의 use case를 작성해야 함

Use case modeling

- 전형적인 진행 절차
  - Find a system boundary
    - 시스템의 내부 요소와 외부 요소를 구분
  - Find the actors
    - 역할(고객, 시스템관리자 등)이 중요
    - 항상 actor는 시스템 외부 요소이지만 그 actor에 대한 내부 표현이 존재할 수 있음. 예) 시스템에서 유지하는 고객 정보
    - Actor는 항상 직접적으로 시스템과 상호작용함
    - Actor의 goal이 무엇인지 분석되어야 함
  - Find the use cases: specify the use case, identify key alternative flows
  - Iterate until stable
- 각 use case와 요구사항 간에 관계도 분석되어야 함

Use case diagram



Use Case Specification

Use case: 판매 세금 지불
ID: 1
간단한 설명: 국세청에 매 분기가 끝날 때마다 판매 수익 세금 지불
주 actor: Time 서비스를 사용하는 주체
보조 actor: 국세청 보통 서비스 제공자
사전조건 1. 분기의 끝
Main Flow 1.매 분기가 끝날 때마다 시작됨 2.시스템은 지불해야 할 총 세금을 계산함 3.시스템은 전자지불방식을 통해 국세청에 세금을 지불함
사후조건 1. 국세청은 올바른 세금을 받았음
Alternative Flow None.

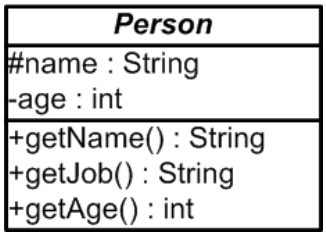
사용자 인터페이스 분석

- 개발하고자 소프트웨어의 사용자가 소프트웨어와 상호작용할 인터페이스를 분석함
  - 어떤 종류의 인터페이스가 언제(어떤 시나리오) 필요하며,
  - 각 인터페이스마다 어떤 정보를 입력 받고, 어떤 정보를 보여주어야 하는지 분석해야 함
- 최종 사용자 인터페이스는 사용자 편리성, 시각적 효과 등을 고려하여야 하지만 초기 분석 과정에서는 인터페이스의 종류, 각 인터페이스마다 필요한 입력과 출력에 대한 검토가 중요함
  - 이를 통해 필요한 데이터의 분석도 가능함
- 가상 인터페이스 디자인의 제시도 바람직함
- 예) 가로세로퀴즈 게임
  - 게임 시작 화면: (입력: 새 게임/기존 게임)
  - 새 게임 시작 화면: (입력: 퀴즈 분야, 난이도)

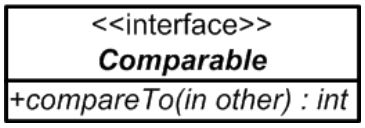
데이터 분석

- 요구사항 분석, 사용 시나리오 분석, 사용자 인터페이스 분석으로부터 어떤 데이터가 언제 필요하며, 어디(유지해야 하는 기간 포함)에 유지되어야 하고, 어떤 과정을 통해 생성, 갱신되는지 분석되어야 함
- 분석의 종류
  - 프로그램 내에 유지해야 하는 데이터
  - 외부 저장공간에 유지해야 하는 데이터
    - 데이터베이스
    - 파일시스템 및 파일
  - 통신 메시지
- 어떤 데이터가 필요한지 분석하는 것이 일차적이지만 이 데이터를 프로그램 내에 어떻게 표현할지 결정하는 것도 필요함
  - 예) 자동 주차장 시스템
    - 필요한 정보: 주차장 크기, 각 주차공간마다 주차여부
    - 표현(자료구조): 층별 boolean 타입 배열(?)

클래스, 인터페이스

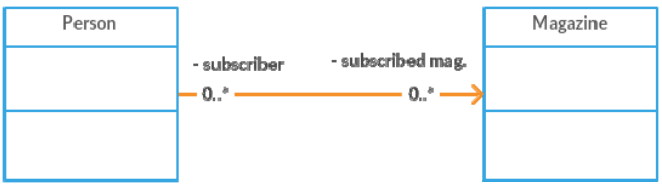


class
첫 행: 클래스 이름 (추상 클래스이면 기울임꼴로 표현)
둘째 행: 객체 상태(멤버 변수)
셋째 행: 객체 행위(메소드), 추상 메소드도 기울임꼴로 표현
- 기호: private
+ 기호: public
# 기호: protected



interface
첫 행: 인터페이스 이름
둘째 행: 행위(메소드)

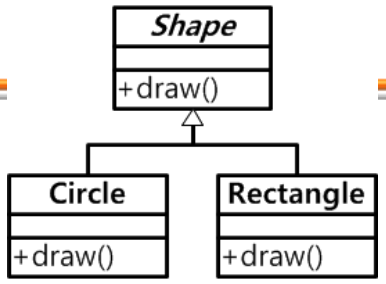
객체 간의 관계



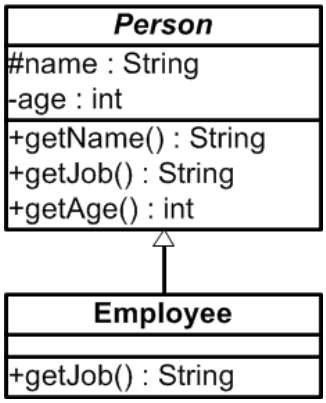
- 인스턴스 수준 관계
  - 사용(use-a): 실선으로 표시
  - 연관(association): 선으로만 표시
  - 집합(aggregation): 빈 다이아몬드로 표시
  - 복합(composition): 진한 다이아몬드로 표시
- 연관 관련 추가 요소
  - 연결 이름: 동사이어야 함
  - 역할
  - 다수 여부(multiplicity)
    - 예) 한 사람은 한 회사에만 고용될 수 있음
    - 예) 회사는 여러 명의 직원을 고용할 수 있음
  - Navigability: 다른 한 쪽은 접근이 가능하지 않음을 나타냄

클래스 간의 관계

- 클래스 수준의 관계
  - 상속(inheritance)
  - 구체화(realization)

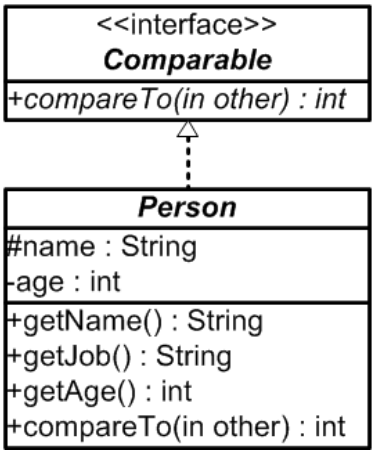


상속



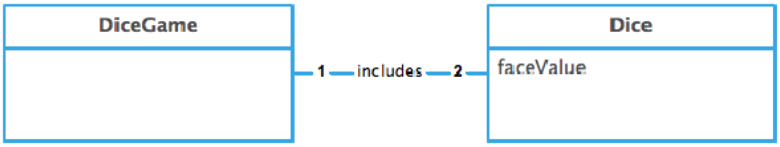
클래스 관계도를 제시할 때에는 이와 같은 세부적인 내용을 포함하지 않음

인터페이스 구현

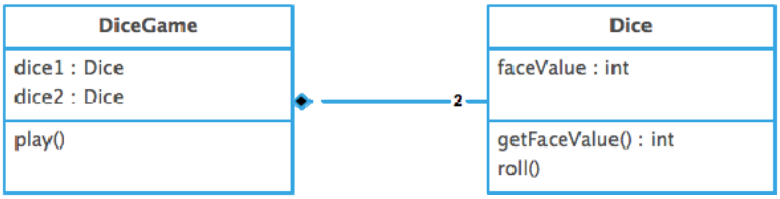


UML 표기법 적용 방법의 종류

- 개념적 측면: 실세계 또는 도메인을 설명하기 위해 작성한 것



- 명세적 측면: 특정 프로그래밍 언어를 고려하지 않고 소프트웨어 설계 측면에서 작성한 것



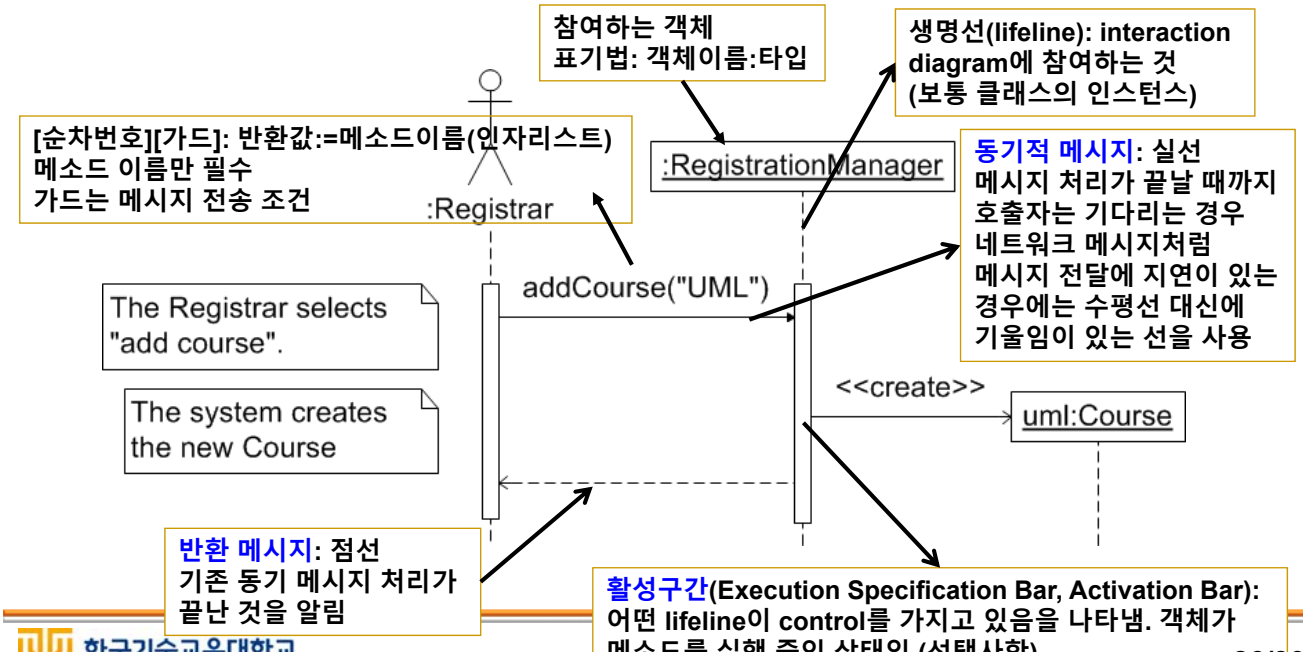
- 구현 측면: 특정 프로그래밍 언어를 고려하여 작성한 것

정적 모델 vs. 동적 모델

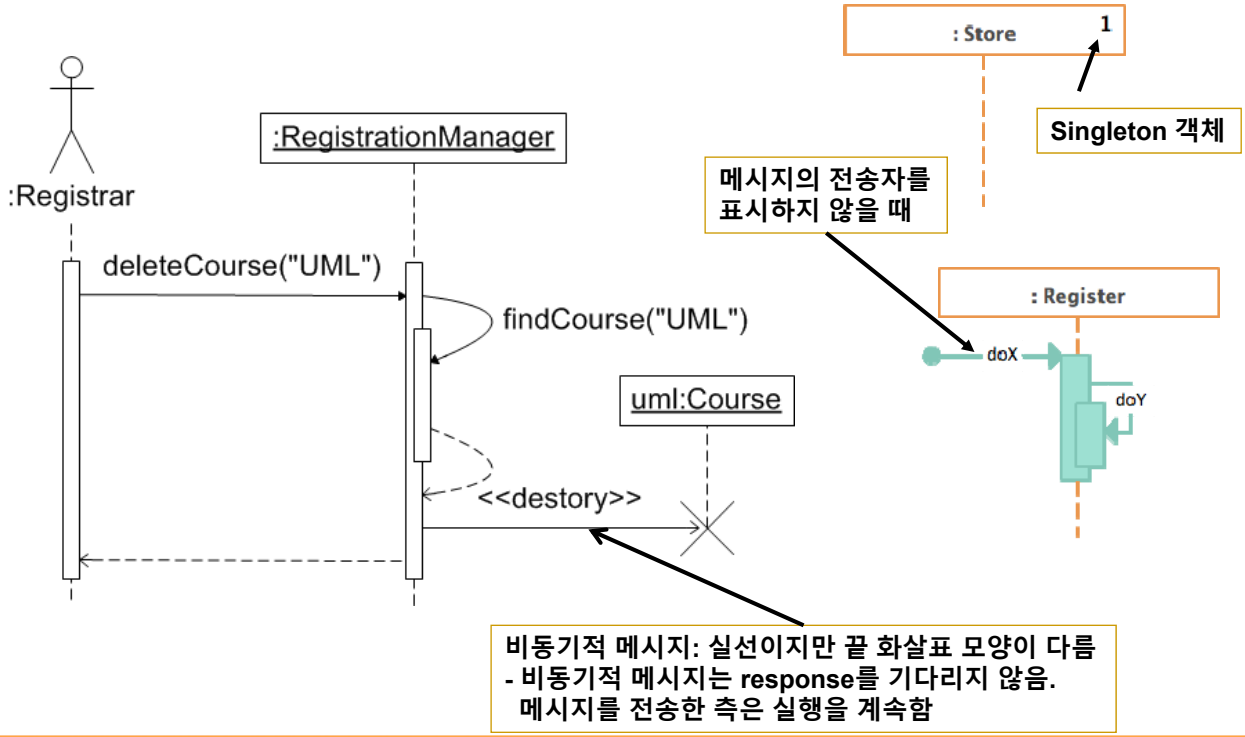
- 클래스 관계도는 주로 클래스 간의 정적 관계를 모델링하기 위해 사용함
- 동적 모델이 중요함에도 불구하고 보통 정적 모델에 더 집중하는 경향도 있음
- 순차 다이어그램(sequence diagram), 통신 다이어그램 (communication diagram) 등 동적 모델링에 충분한 시간을 투자할 필요가 있음

시퀀스 다이어그램 (1/4)

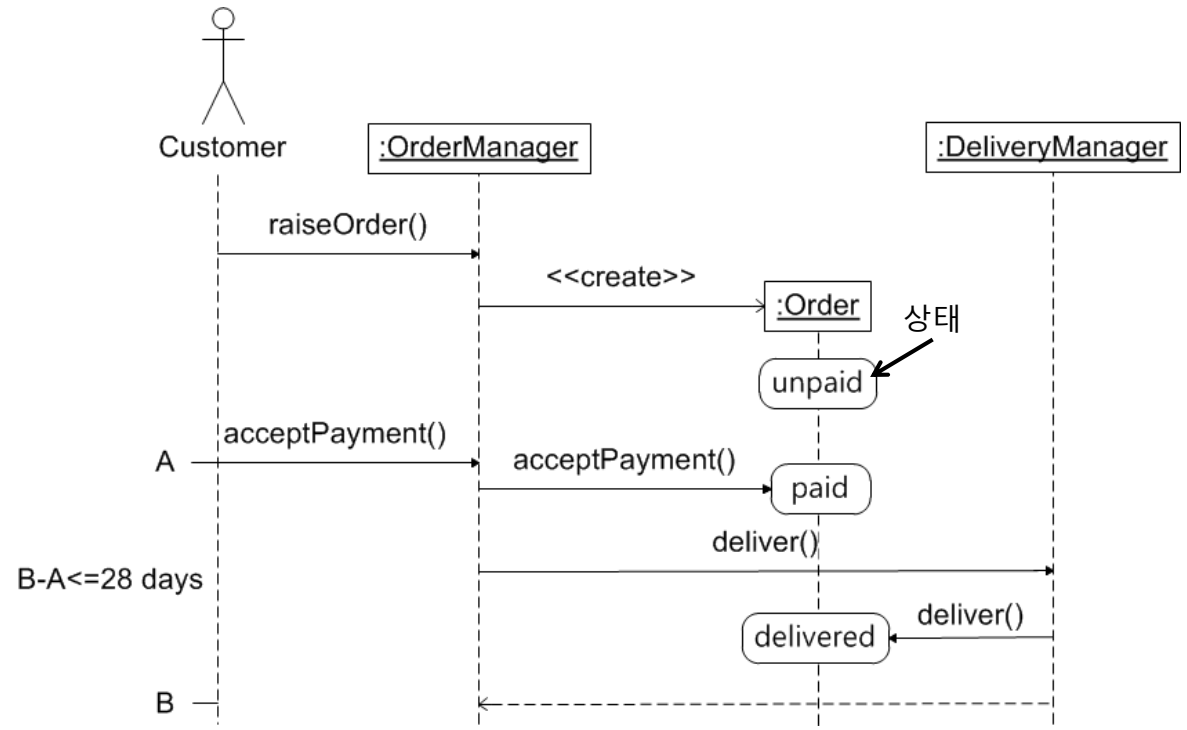
- 순차 다이어그램: 시간 흐름에 따른 객체들간의 상호작용 순서를 보여줌
  - 도출된 클래스들에서 생성된 객체들이 어떤 서비스를 제공하기 위해 어떤 순서로 상호작용하는지 보여줌



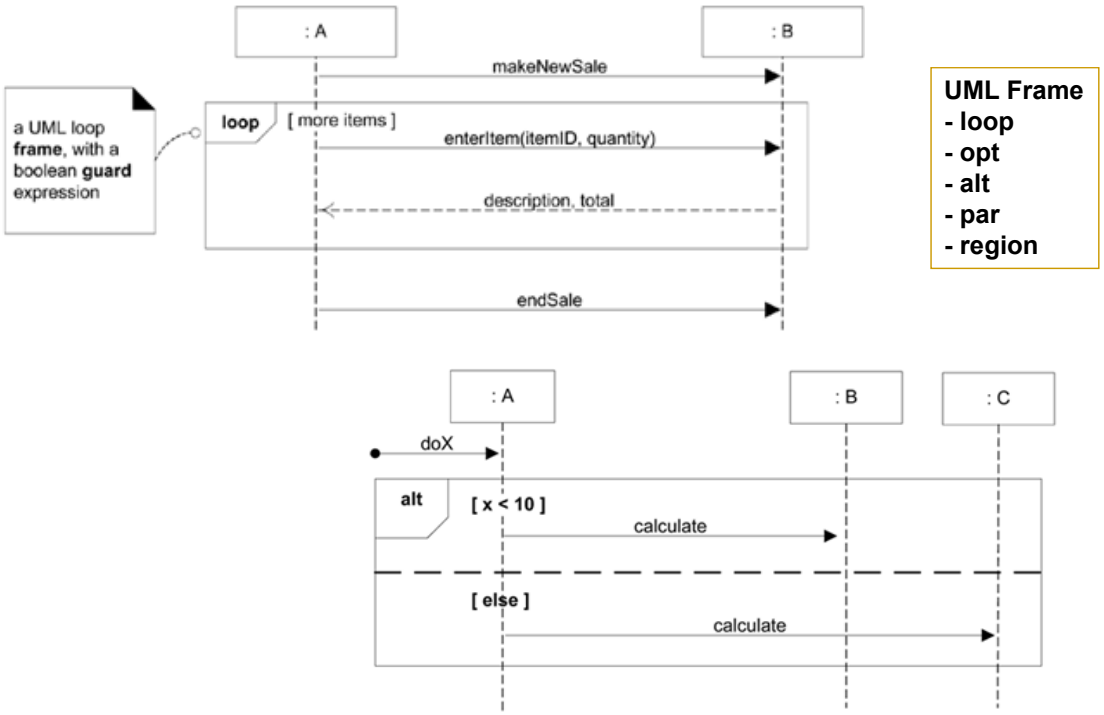
시퀀스 다이어그램 (2/4)



시퀀스 다이어그램 (3/4)



시퀀스 다이어그램 (4/4)



구현

- 테스트 기반 프로그래밍
  - 메소드 정의 전에 단위 검사 코드부터 작성
- 예외 처리, 오류 처리 전략 수립
- 클래스 관계도를 이용한 클래스 정의
- 시퀀스 다이어그램을 이용한 메소드 정의
- 짝 프로그래밍
- 리펙토링
  - 테스트 기반 활용