Introduction to OOAD using UML tools

200611450 강세용
200610118 김규수
201060682 Valentin

이제부터 OOA/D에 관하여, UML을 사용한 소프트웨어의 개발이 어떻게 이루어지는지를 알아보도록 하겠습니다. 일단은 전반적인 소프트웨어 개발 순서와 각각의 산출물에 대해서 언급하고, 마지막에 각각의 다이어그램 별로 그 다이어그램들을 어떻게 사용하는지에 대하여 설명하겠습니다.

지금 소개해 드릴 것은 GRAPPLE이라는 "Guidelines for Rapid APPLication Engineering“ 이라는 것이며 이것은 RUP의 아이디어와 많이 흡사한 UML을 소프트웨어 개발 사례에 적용하는 지침중의 하나입니다.

GRAPPLE

GRAPPLE은 다섯 개의 진행영역(Segment)으로 이루어져있다.
각 진행 영역은 순서대로 여러개의 동작(action)으로 구성되어 있으며 각각의 동작에서 작업 결과물(Work-product)가 만들어지며, 각 동작에 대한 담당자가 구분되어 있다. 대부분의 경우, 이렇게 만들어진 작업 결과물은 프로젝트 관리자에 의해서 종합되어 의뢰인에게 보여지지만, 사실 작업 결과물은 단지 의뢰인에게 보이기 위해서 만드는 것이 아니라, 불필요한 문서화를 따로 하지 않고 작업 과정의 진척을 유지하고 관리하기 위해서 만든다.

GRAPPLE은 객체지향 시스템을 기준으로 고안되었으며, 각 진행 영역을 이루고 있는 동작들은 객체지향적 성격에 맞는 작업 결과물을 만들어내도록 조정되었다.

다음이 다섯 개의 진행영역이다.

1. 요구사항수집(Requirement Gathering)
2. 분석(Analysis)
3. 설계(Design)
4. 개발(Development)
5. 배포(Deployment)

위의 앞머리를 하나씩 따면 RADDD가 되는데, 이것을 줄여서 RAD^3이라고 부른다.

1. 요구사항수집(Requirement Gathering)

다섯 개의 영역중에 가장 중요한것은 바로 요구사항 수집이다. stackholder가 무엇을 원하는지도 모르는데, 제대로된 시스템을 만들 수는 없다. 제대로된 요구사항을 이해하기위해서는 도메인의 핵심을 이해해야한다.

1.1 업무과정 파악

훌륭한 개발 작업은 의뢰인의 업무가 어떻게 진행되는지를 이해하는 것부터 시작된다. 분석가는 의뢰인(stakholder) 혹은 의뢰인이 인정한 "관계자"를 만나서 해당 업무가 어떤 단계로 진행되는지에대한 이야기를 나눈다. 이곳에서 도메인 영영의 용어들을 얻을 수 있으며, 분석가는 차후에 이 어휘를 사용해 의뢰인과 이야기를 한다.

이때 만들어지는 작업 결과물은 해당 업무과정에서 단계와 결정 위치를 정리하여 나타낸 활동 다이어그램(Activity Diagram)이 된다.

1.2 도메인분석

이 동작은 의뢰인(stackholder)나 의뢰인이 인정한 관계자와 이야기를 나눌때 수행되며, 의뢰인의 도메인을 가능한 더욱 탄탄하게 파악하가 위함이 목적이다. 이 동작은 앞의 업무과정파악과 비슷한 개념적인 성격을 가지고, 구축하고자 하는 구체적인 시스템 사항은 아직 나타나지 않는다. 분석과는 이 과정에서 도메인에 관련된 최대한 많은것을 알아내고 분석해야한다.

객체 모델러는 대화중의 명사를 정리하여 클래스를 만들기 시작하며, 어떤 명사는 속성, 어떤 동사는 오퍼레이션등으로 분류 할 수 있을 것이다. 이 작업은 모델링 도구를 사용하는 것이 능률적이며, **이 문서의 UML도구는 starUML을 사용하여, 대부분의 다이어그램모형과 관계도를 작성하였다.**

이 동작에서 만들어지는 작업 결과물은 추상적인 수준(high-level)의 클래스 다이어그램과 대화내용을 정리한 노트가 되겠다.

1.3 연동 시스템 확인

이 동작의 초기에 개발팀은 자신들이 만든 새로운 시스템이 의존할 시스템과 이 시스템과 잘 맞물리는 시스템을 찾는다. 시스템 엔지니어는 이 동작의 진행과정을 예의 주시한 다음, 배포 다이어그램을 작업 결과물로 만들어 낸다.

1.4 시스템요구사항 확인

이 동작에서 개발팀은 처음으로 공동 어플리케이션 개발(Joint Application Development)JAD를 가지게 될 것이며 이 모임에는 의뢰인(stackholder)이 속한 그룹의 의사결정권자와 잠재 사용자, 개발팀 맴버들등이 참여한다.

이곳에서 개발팀은 의견을 조율하고, 객체 모델러는 앞서 그려진(도메인분석)클래스 다이어그램을 손질한다.
여기서 만들어지는 작업 결과물은 패키지 다이어그램이며, 이곳에 추상적인 수준의 시스템 기능이 담겨있다. 그리고 각각의 패키지에는 그룹화된 유스케이스들도 포함될 것이다.

2. 분석(Analysis)

이 진행 영역은 “요구사항 수집” 등의 결과를 가지고 더욱 자세히 작업하여 문제의 이해도를 높인다. 이 영역은 독립적으로 시행되는게 아니라 전 단계인 요구사항 분석단계중에 시작되는데, 보통은 JAD모임의 클래스 다이어그램을 손질할 때 이 영역이 같이 시작된다.

2.1 시스템 사용법의 이해

이 동작은 추상 수준의 유스케이스 분석작업이며 개발팀은 각 유스케이스를 시작하게되는 행위자와 그 유스케이스의 결과를 얻는 행위자를 알아낸다. 행위자는 사람이 될수도 시스템이 될 수도 있다. 또한 유스케이스를 새로만들어내고, 추상 유스케이스도 만든다. 이 동작에서 만들어지는 작업 결과물은 행위자와 유스케이스간의 스테레오타입을 가진 의존관계를 나타내는 유스케이스 다이어그램이다.

2.2 클래스다이어그램 손질

JAD모임이 진행되는 동안 객체 모델러는 회의내용을 종합하여 다시 클래스 다이어그램을 계속하여 고쳐나간다. 이 때, 객체 모델러는 연관, 추상클래스, 다중성, 일반화, 집합연관등의 내용을 모두 적어나가며 여기서 만들어지는 작업물은 정형화된 클래스 다이어그램이다.

2.3 객체의 상태 변화 분석

객체 모델러는 상태 변화를 나타냄으로써 모델을 더 자세하게 만든다. 작업 결과는 상태 다이어그램이다.

2.4 객체간 교류 정의

개발팀은 유스케이스 다이어그램과 정형화된 클래스 다이어그램들을 가지고, 객체들이 어떻게교류하는지를 정의해야한다. 객체 모델러는 시퀀스 다이어그램과 협력 다이어그램을 만든다. 작업겨로가물은 시퀀스 다이어그램, 협력 다이어그램이다.

3. 설계(Design)

이 진행 영역은 “분석” 진행 영역에서 얻어낸 결과를 가지고 솔루션을 설계한다. “분석”과 “설계”도 서로 완전한 독립적인 진행영역이 아닌, 설계가 완료될때까지 서로의 영역을 자유롭게 넘나들 수 있다.

3.1 객체 다이어그램의 개발과 손질

프로그래머는 클래스 다이어그램을 가지고 필요한 객체 다이어그램을 그린다. 각각의 오퍼레이션을 체크하고 여기에 맞는 활동 다이어그램을 그림으로써, 완전한 객체 다이어그림을 그린다. 활동 다이어그램은 “개발”진행영역의 대부분의 코딩작업 기반을 제공하며, 이 영역의 작업 결과물은 객체 다이어그램과 활동 다이어그램이다.

3.2 컴포넌트 다이어그램의 개발

프로그래머가 재대로된 활동을 할 수 있는 단계가 이 부분이다. 여기서 할 일은 컴포넌트를 시각화 하고 각각의 의존관계를 나타내는 것이다. 작업 결과물은 컴포넌트 다이어그램이다.

3.3 배포계획

컴포넌트 다이어그램이 만들어지고 나면, 시스템 엔지니어는 시스템 배포와 연동 시스템의 통합을 계획한다. 시스템 엔지니어는 배포 다이어그램에 컴포넌트를 적적히 배포해 그려넣는다. 여기서 만들어지는 결과물은 이전에 그린 배포다이어그램의 일부가 될 부분 다이어그램이다.

3.4 사용자 인터페이스의 설계와 원형 정의

사용자 인터페이스는 모든 유스 케이스를 고려하여 만들어져야 한다. GUI분석가느 ㄴ사용자와 함께 각각의 유스케이스에 맞는 화면 프로토타입을 그리고, 사용자는 화면 컴포넌트를 원하는 위치에 붙인다. 각 컴포넌트의 위치에 만족하면, 개발자는 사용자 인터페이스의 원형을 정의할 수 있게 된다. 여기서 작업결과물은 화면 원형을 잡은 스크린샷이다.

3.5 시험설계

유스 케이스는 소프트웨어를 시험하는데 설계의 용도로 쓰인다. 개발된 소프트웨어가 유스케이스에서 설정한대로 잘 동작하는지 평가하기 위해서이다. 이 과정에서 유스케이스 다이어그램을 사용하여 시험용 스크립트나 자동화된 시험도구를 개발한다. 결과물은 시험 스크립트가 된다.

3.6 문서화 시작

최종 사용자나 시스템 관리자를 위한 시스템 설명서를 작성하기 시작한다. 문서화는 시스템 설계자와 함께 스토리 보드를 작성하기 시작하고 문서의 골격을 만든다. 결과물은 문서의 골격이다.

4. 개발(Development)

개발은 프로그래머가 직접적으로 나서며, 이미 충분한 분석, 설계작업을 거쳤기 때문에 개발작업은 빠르고 부드럽게 진행한다.

4.1 코드작성

프로그래머는 완성된 클래스 다이어그램, 객체 다이어그램, 활동 다이어그램, 컴포넌트 다이어그램을 가지고 시스템에 필요한 코딩을 시작한다. 작업 결과물은 코드이다.

4.2 코드시험

완성된 코드가 원하는 대로 작도앟는지 시험 스크립트를 사용하여 평가한다. 해당 코드가 모든 시험을 통과할때까지 "코드 작성" 단계에 반영하여 피드백으로 사용할 수 있으며, 그 반대도 가능하다.

4.3 사용자 인터페이스의 구축과 코드의 연결 및 시험

사용자의 확인을 거친 사용자 인터페이스를 직접 구현한다.

4.5 문서화 완료

프로그래머의 작업과 동시에 진행하며 소프트웨어 개발 완료와 동시에 끝난다. 결과물은 시스템에 관련된 설명서이다.

5. 배포(Deployment)

배포의 영역은 "개발" 진행이 시작하기 전에 이루어진다. 이 영역에서 소프트웨어는 다른 시스템과 연동되며, 백업 복구에 관한계획등 시스템 설치 시험으로 모든 영역이 종료된다.

여기까지가 Grapple에 대한 전반적인 진행 과정이며, 대부분의 소프트웨어는 이와 비슷하게 만들어집니다. 특히 분석과 설계에 진행되는 각각의 다이어그램들과 유스케이스 같은 문서 산출물 들은 객체지향 분석과 설계를 더욱 확실하게 만들어주며 이것들은 반복되어 진행될수록 더 정확하게 요구사항에 맞춰 시스템을 구현할 수 있습니다.

이제부터 각각 단계 중, 분석과 설계 단계에 쓰인 다양한 다이어그램들은 상세하게 사용하는 방법을 설명하겠습니다. 또한 이것들에 의해 어떻게 시스템이 객체화되고 모델링 되는지도 알아보겠습니다.

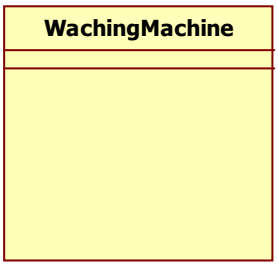
UML Diagram

UML Diagram의 종류는 다음과 같이 분류된다.

Step	Process	Diagram
Step1	Static Aspect	Class/Object Diagram
Step2	Functional Aspect	Usecase Diagram
		Collaboration Diagram
		Sequence Diagram
Step3	Dynamic Aspect	Activity Diagram
		StateDiagram
Step4	Physical Aspect	Component Diagram
		Deployment Diagram

클래스/객체 다이어그램 (Class/Object Diagram)

1.클래스 표현의 기본
클래스를 UML로 나타낼 때는 사각형 안에다가 클래스 이름을 적어 넣는다. 이때 클래스 이름은 첫 자를 대문자로 적고, 사각형의 상단에 가깝게 두는 것이 보통이다. 클래스 이름이 두 단어로 되어있다면 두 단어를 붙여 쓰고, 둘째 단어의 첫 자도 대문자로 쓴다.



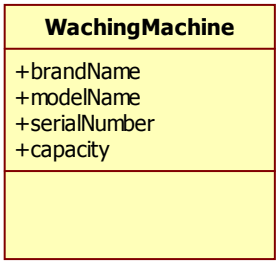
만약 클래스가 다른 패키지 안에 속해있다면,

패키지이름::클래스 이름

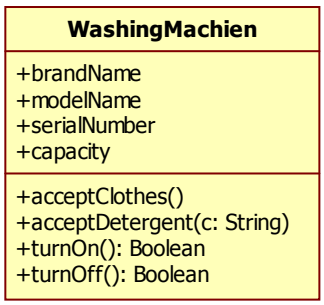
이런 식으로 표현한다. 이렇게 쓴 클래스를 경로이름(pathName)이라고 부른다.

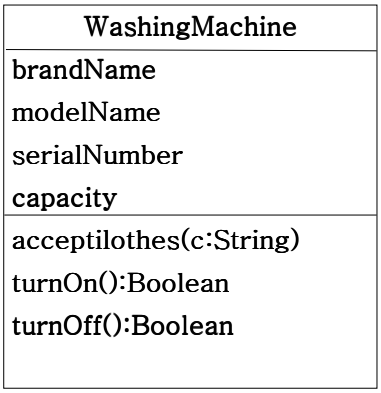
1.1.속성(Attribute)
클래스에 속한 특성에 이름을 붙인 것이다.
클래스는 0개 이상의 속성을 가질 수 있으며, 한단어로 된 속성의 이름은 소문자로 쓰는 것이 보통이며, 속성 이름이 두 단어 이상으로 되어있다면, 두 단어를 붙여쓰되, 두 번째 단어의 첫 문자를 대문자를 쓴다.

1.2.오퍼레이션(operation)
객체에게 요청할 수 있는 행동을 말한다. 속성이름과 마찬가지로 소문자 시작하며, 둘째 단어부터는 첫 자를 대문자로 시작한다. 속성(Attribute)밑에 구분선을 그어 그 밑에 위

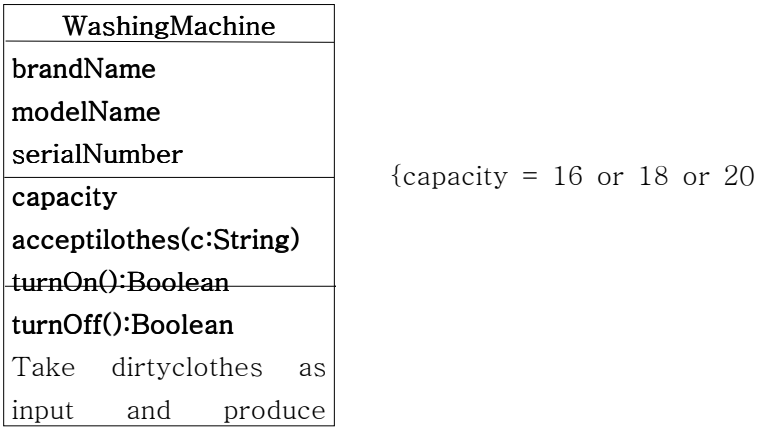


치시킨다. 또한 매개변수(parameter)리스트를 넣어줄 수 있으며 매개변수는 “이름:타입" 의 형태를 가진다. 반환값은 괄호 뒤에 콜론을 찍고, 값과 타입을 적으면 된다.
이러한 오퍼레이션의 정보를 오퍼레이션의 시그니처(signature)라고 한다.



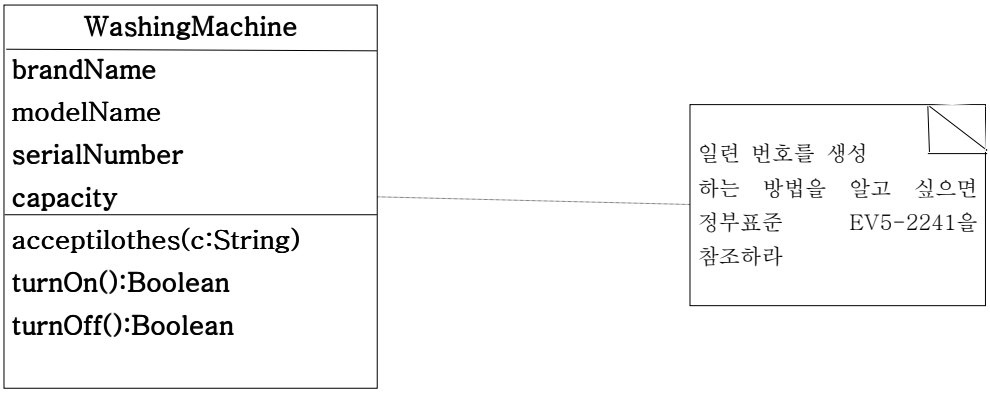


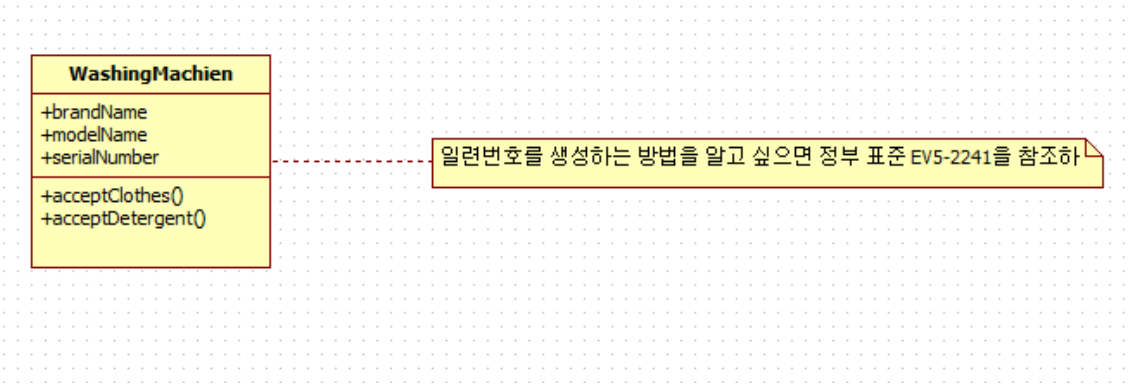
1.3.책임(Responsibility) 및 제약(Constraints)
클래스 아이콘에는 위의 표기법 외에도 책임(responsibility)를 부여할 수 있다. 책임이란, 특정한 타입 혹은 클래스가 “해야 하는” 일을 설명해 둔 것이다.



표기법은 위와 같다. 또한 오른쪽의 대괄호는 속성이 가지는 제약(constraints)다. 즉 위 wasingMachine의 capacity는 16,18,20세가지 값중 하나만 가질 수 있다.

1.4.노트(Node)
속성, 오퍼레이션, 책임, 제약 이외에도 클래스에 추가적인 정보를 덧붙일 수 있는 수단이 하나 더 있는데, 노트(note)라고 불리는 것이다.





노트는 텍스트와 그림을 동시에 사용할 수 있다.

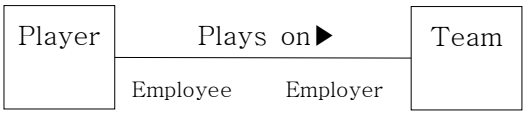
2.클래스 모델링의 시작

클래스는 지식 도메인에 기반한 어휘와 용어로부터 만들어진다. 시스템 분석가는 의뢰인과 상담하여 그들이 가지고 있는 지식 도메인을 파악하여 정리하고, 그 도메인에서 발생되는 무제를 해결할 컴퓨터 시스템을 설계해 나가면서, UML에 사용할 용어를 선정하고 이것을 클래스로 모델링하는 것이다.

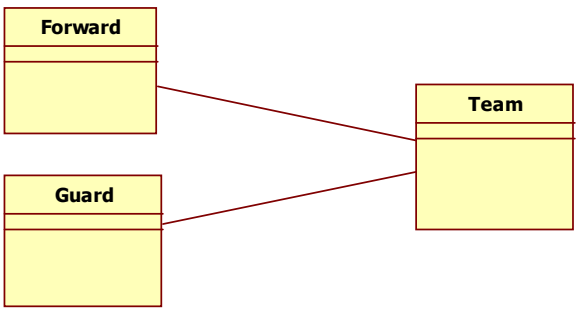
2.1클래스 관계

2.1.1.연관(Association)
클래스가 개념적으로 연결되어있을 때, 이 관계를 연관이라고 부른다. 그리고 이것을 표시하기 위해선. 두 클래스를 선으로 잇고, 선 위에 연관 이름을 붙인다. 둘 사이의 관계를 방향을 지정해 줄 수 있는데, 채워진 화살표 머리를 붙여주면 된다.
이것은 팀에 속한 선수의 연관을 그림으로 나타낸 것이다.

또한 한 클래스가 다른 클래스와 연관이 되면, 각각은 해당 연관관계 내에서 역할을 가진다. 클래스 옆에다가 원하는 역할을 써 줌으로 써 연관관계 내에서의 여고할을 표시할 수 있다. 즉 아래 그림처럼, Player와 Team간의 고용주, 피고용주 역할이 성립할 수 있는 것이다.



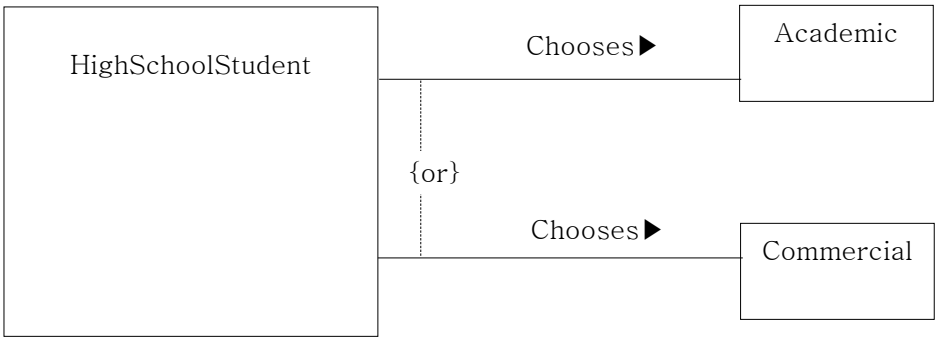
연관은 하나의 클래스가 다른 클래스에 연결 된 것 이상으로 복잡해질 수도 있다.
아래 그림처럼 여러 개의 클래스가 하나의 클래스에 연결되는 것도 가능하며, 문어발처럼, 하나의 클래스에서 여러 클래스로 연결되는 것도 가능하다.



2.1.2.연관에 대한 제약

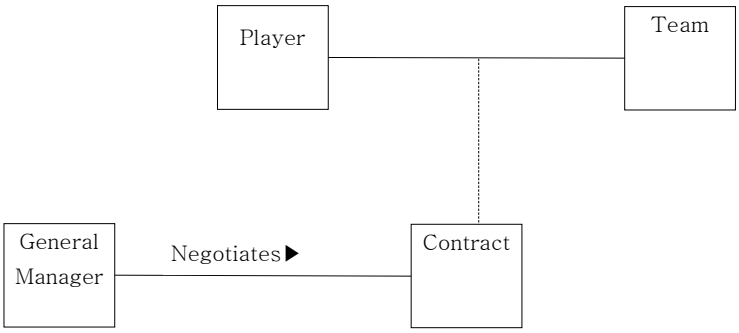
두 클래스 사이의 연관관계가 어떠한 규칙을 따라야 할 경우가 있다. 이 규칙을 덧붙일 수 있는데, 앞의 클래스에서 알아본 제약(constraint)을 연결선 부근에다가 쓰면 된다. 예를 들어, 은행원(Bankteller)은 고객(Customer)의 은행 업무를 도와주지만(Serves), 각 고객들이 창구에서 나온 순서(Order)대로 은행 업무를 도와준다. 이 상황을 모델링하면 아래처럼 그릴 수 있다. ordered란, 텍스트가 담긴 중괄호(제약)를 Customer클래스 가까이 붙이는 것이다.

또 한 가지의 제약ㅇ은 두 개의 연관선 사이를 점선으로 잇고 이 위에 {or}로 표기하는 Or 관계이다. 아래 그림은 고등학생들이 진로를 정하는데(choose) 있어서 학문적인(academic)코스 또는 상업적인(commercial)길을 선택할 수 있다는 상황을 모델링한 것이다.



2.1.3.연관 클래스

연관은 클래스와 같이 속성과 오퍼레이션을 가질 수 있다. 사실, 속성과 오퍼레이션을 가진 연관은 연관클래스(Association class)라는 이름으로 따ㅣ로 구분한다. 연관 클래스를 그리는 방법은 클래스를 그리는 방법과 동이랗며, 연관선과 이으려면 점선을 사용한다. 연관 클래스도 다른 클래스에 대하여 연관을 가질 수 있다.



위 그림은 Player와 Team 사이의 plays on연관에 대한 연관클래스를 보이고 있다. 여기서 연관클래스는 Contract이며, 이 연관클래스는 GeneraManager와도 연관이 있다.

2.1.4.링크(Link)

객체가 클래스의 인스턴스인 것처럼, 연관도 자신의 인스턴스를 가질 수 있다. 어떤 특정한 선수가 특정한 팀에 소속되어 있는 관계를 생각하면, 이 때의 Plays on 연관 관계를 링크(link) 라고 부른다. 링크는 두 객체(인스턴스)를 선으로 이어서 나타내며, 객체에 밑줄을 긋듯이 링크에도 밑줄을 긋는다.

2.1.5.다중성(Multiplicity)

Player와 Team이 가지고 있던 지금까지의 연관 관계는 일대 일 관계였다. 즉 하나의 클래스에 하나의 클래스가 연관되어 있는 상태다. 하지만 클래스간의 관계는 일대 다수 또는 다수대 하나 일 수 있으며 이런 모든 것을 고려할 수 있어야한다. 이것을 다중성(Multiplicity)이고 한다.

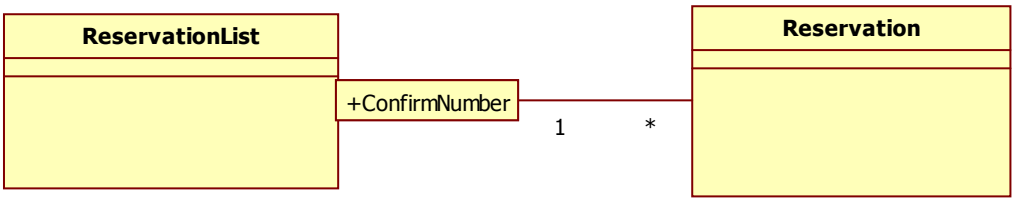
위 그림이 5개의 StartingPlayer를 하나의 Team에 관련시키는 방법이다.
다중성은 일 대 일(one-to-one), 일 대 다 (one-to-many) , 일 대 일 또는 그 이상(one-to-one or more), 일 대 0 또는 1(one-to-zero or one), 일 대 일정 범위 (one-to-a bounded interval) 등등 다양하게 표현 할 수 있다.

UML은 “more" ”many"를 표현하는 기호로써 *를 사용한다.

2.1.6.수식연관

일 대 다 (one-to-many)의 다중성을 가지 ㄴ연관관계에서는 한 객체가 특정한 객체를 가려내어야 하는 상황이 발생한다.(lookup)한쪽 클래스의 객체가 다른 쪽 클래스의 객체를 선택하여 연관관계 내에서의 역할을 만족시켜야 할때 첫째 클래스는 자신이 원하는 객체를 찾끼 위해 특정한 속성에 의존할 수 밖에 없는데, 대개 특정한 속성은 ID 번호 같은 식별자에 해당한다.

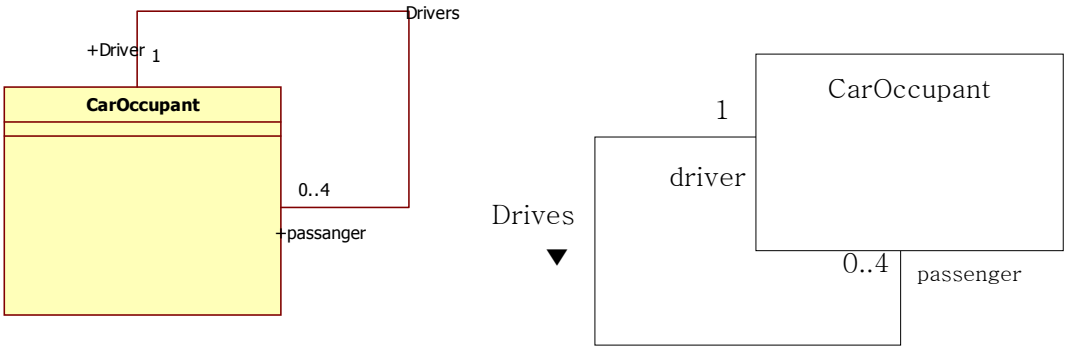
위 그림과 같이 예약 리스트와 연결된 다수의 예약들 중 특정한 확인 번호를 사용해 예약을 찾고 싶을대, UML에서 이런 특정한 확인번호를 수식자(qualifier)라고 하여 작은 사각형으로 나타내고 일 대 다수의 다중성에서는 "1"을 의미하는 클래스의 옆에 붙는다.



그래서 위 그림처럼 수식자- confirmationNumber 와 Reservation 클래스 사이는 일 대 일의 관계가 성립한다.

2.1.7.반사연관(Reflexive association)

클래스는 자신과 연관관계를 가질수도 있으며 이것을 반사연관이라고 한다. 여러 가지 역할을 맡을 수 있는 객체를 가지고 있을대, 이런 경우가 발생하며, 예를들어 탑승자(CarOccupant)는 운전자(Driver)도 될 수 있고, 승객(Passenger)도 될 수 있다. 이것을 UML로 나타내려면 연관선이 자신의 클래스 사각형을 가리키도록 긋고, 연관선에다가 각각의 역할과 연관의 이름, 연관의 방향 그리고 다중성 관계를 표시해주면 된다.



위 그림은 사각형 두 개가 겹쳐있는 것이아니라, CarOccupant클래스에서 출발한 연관선이 자기 자신에게로 다시 돌아오는 것이다. 연관은 Drives오 연결되어있으며, 한명의 driver는 0에서 4까지의 passenger와 관계를 가질 수 있다.

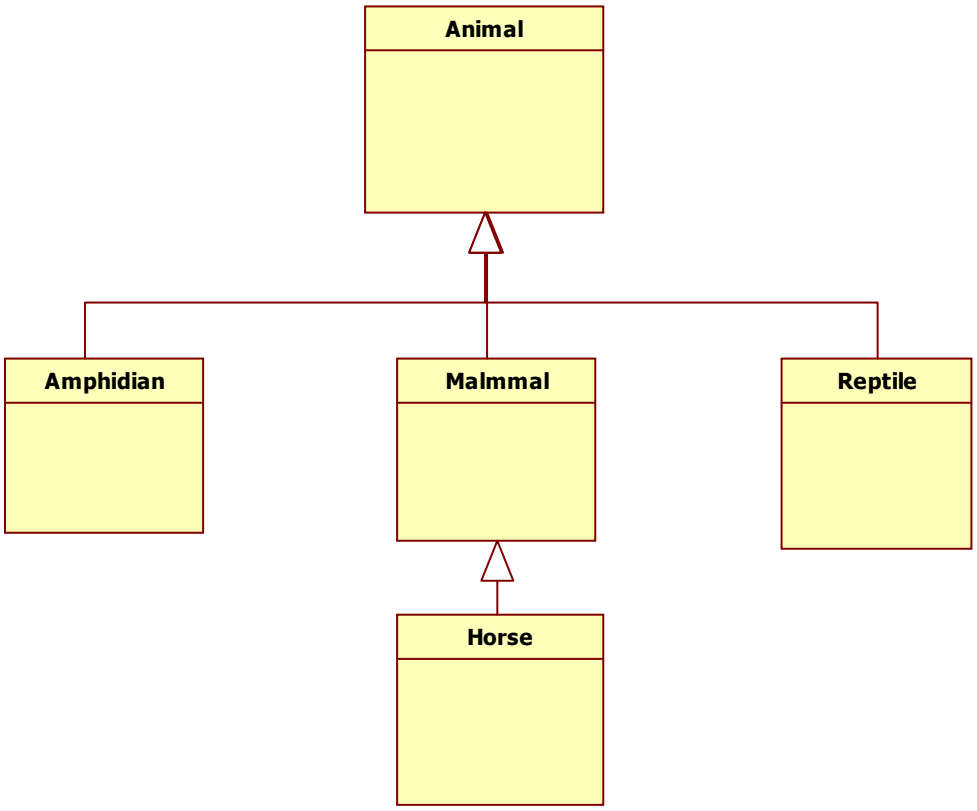
2.2.상속과 일반화(Inheritance and Generalization)

객체지향은 “일상 생활의 보편적인 특징을 멋지게 표현해 낼 수 있다는 ” 점에서 우수하다. 어떤 사물이 속한 범주를 알고 있으면, 이 범주에 속해있는 다른 사물의 특성을 알 수 있다.

객체지향 개념에서는 이것을 상속(Inheritance)라고 하며 UML에서는 일반화(Generalization)이라고 한다. 한 클래스는 다른 클래스로부터 속성과 오퍼레이션을 물려받을 수 있으며, 슈퍼클래스는 서브클래스보다 일반적인 특성을 가진다.

UML에서 클래스의 상속관계를 나타낼 때에는 서브클래스에서 슈퍼클래스로 선을 그은

다음, 슈퍼클래스 쪽에다가 속이 빈 화살표 머리를 붙여준다. 이런 타입의 연결관계를 "....의 일종" 이라고 부른다.



위 그림처럼 하얀색 빈 삼각형의 방향으로 상속성을 결정한다. 서브클래스는 슈퍼클래스로부터 상속받은 속성과 오퍼레이션 이외에 자신만의 것을 더 추가할 수도 있다. 슈퍼클래스를 가지지 않는 클래스는 기본클래스(Base Class)또는 루트클래스(root class)라고 하며 비슷하게 서브클래스를 가지지 않는 클래스를 리프 클래스(leaf class)라고 한다. 한편 클래스가 하나의 슈퍼클래스를 가지면 단일상속(single inheritance)라고하며 두 개 이상의 슈퍼클래스를 가지는 상속은 다중상속(multiple inheritance)라고 부른다.

2.2.1.상속관계의 정의

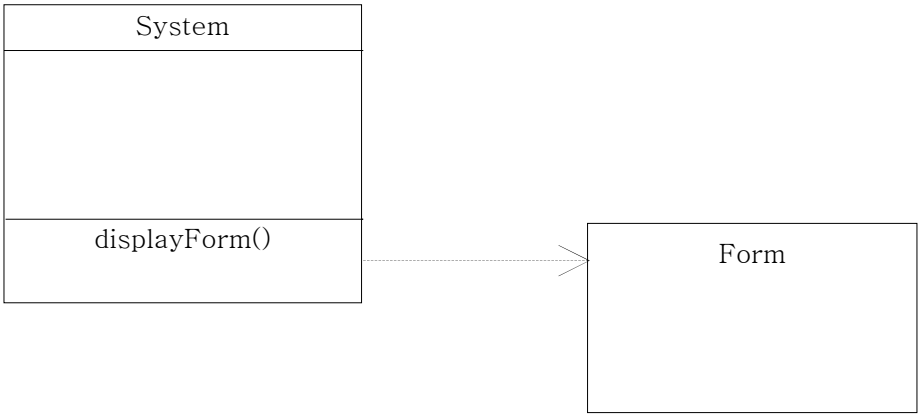
시스템 분석가는 고객과의 상담과정에서 여러 가지 상속관계를 발견해낸다.
시스템 분석가는 한 클래스에 속해 있는 속성과 오퍼레이션의 일반성을 찾아내어 다른클래스에 적용할 수 있는지를 재빨리 간파해야한다.

2.2.2.추상클래스

또한 클래스 중에는 인스턴스를 지원하지 않는 클래스가 있을 수도 있다. 이런 클래스를 추상클래스라고 하며 이탤릭체로 쓴다.

2.2.3.의존관계(dependency)

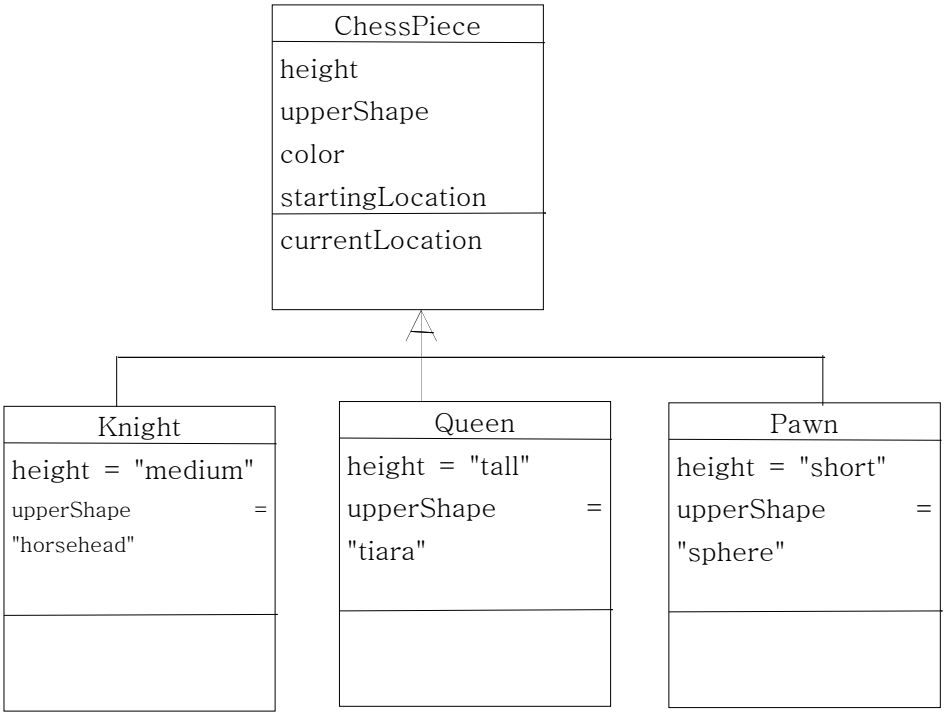
“한 클래스가 다른 클래스를 사용하는” 관계가 하나 더 있다. 이것을 의존관계(dependency)라고 한다. 의존관계가 가장 흔하게 드러난 예는 다른 클래스를 사용하는 오퍼레이션의 시그니쳐를 보일때이다.
예를들어 직원들이 써 넣을 수 있는 Form을 표시해 주는 시스템을 설계한다고 할때, 직원들은 서식을 선택하고, 이 서식에 내용을 적는다. 만약 이 서식에 관련된 내용을 출력한다고한다면, System의 출력은 사용자가 어떤 서식을 선택했냐에따라 전적으로 의존할 수 밖에 없다. 이 상황을 UML로 표시하려면 아래와 같다.



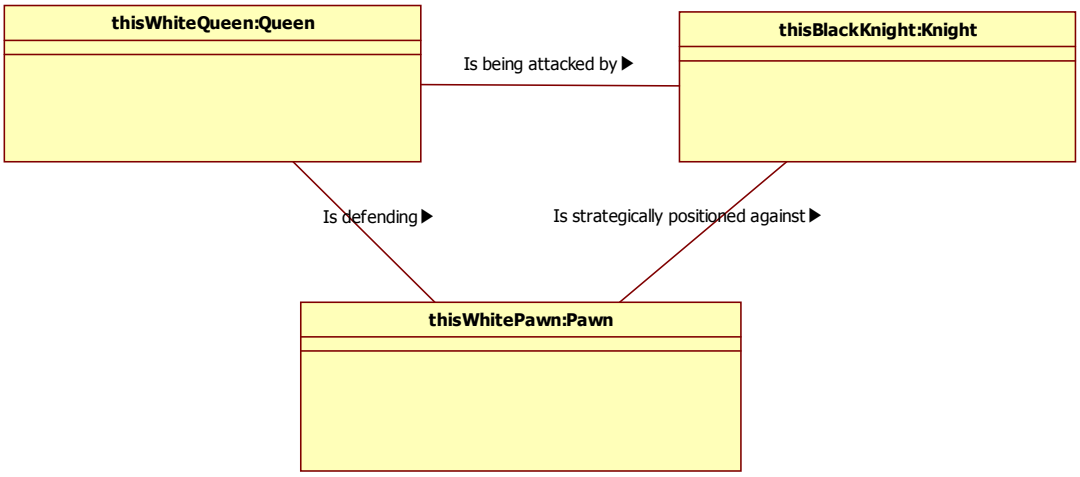
System의 displayForm()은 Form과 의존관계를 갖는다.

2.3.클래스 다이어그램과 객체 다이어그램

클래스 다이어그램은 일반적인 정의성(definitional) 정보 를 알려준다. 반면에, 객체 다이어그램은 클래스의 특정한 인스턴스 정보와 어떠한 상황에 처해 있는지의 정보를 보여준다. 시간의 개념(instants)과 인스턴스(instances)는 객체 다이어그램의 개념을 잘 설명할 수 있는 단어가 되겠다. 아래 그림은 체스게임을 클래스 다이어그램으로 나타낸 것이다.



그리고 이 그림이 객체 다이어그램이다.

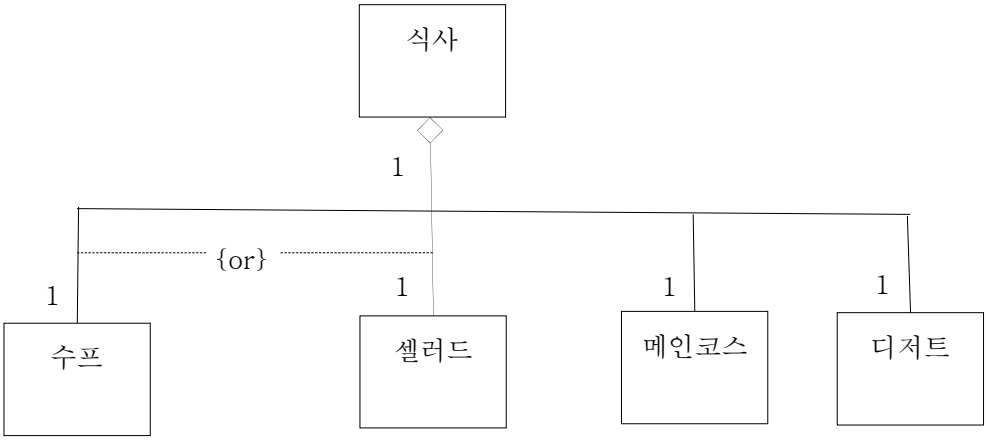


클래스 다이어그램으로는 부족한, 위치정보, 말의 공격 대상등을 객체 다이어그램으로 해결 할 수 있다.

집합연관, 복합연관, 인터페이스 그리고 실체화

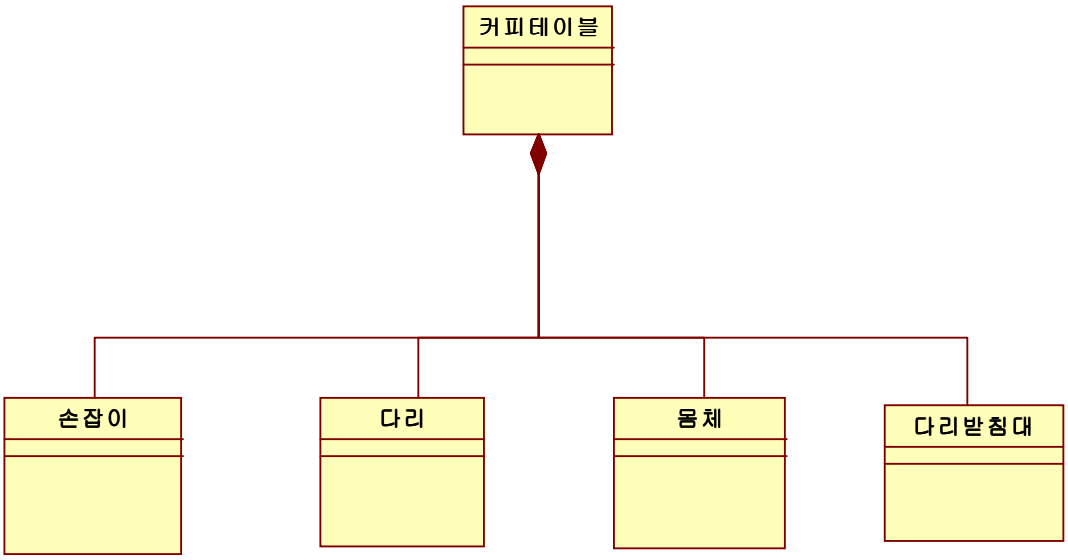
2.4.집합연관(aggregation)
하나의 클래스가 여러개의 컴포넌트 클래스로 구성되어 있는 경우, 이러한 경우를 집합연관이라고 한다. 컴포넌트 클래스와 전체 클래스는 부분-전체(part-whole)연관관계를 가진다.

집합 연관은 계층도로 나타내며 “전체(whole)"클래스는 윗부분에 위치하며, 컴포넌트 클래스는 아랫 부분에 위치한다. 전체 클래스와 컴포넌트 클래스는 선으로 연결되며, 전체 클래스 쪽에 빈 마름모꼴이 붙는다.
집합연관의 컴포넌트 들은 Or관계에 놓일 때도 있으며 이것을 모델링하려면 사이에 제약을 두면된다. 두 개의 집합연관 선 사이를 점선으로 이은 다음에 {or}를 써준다.



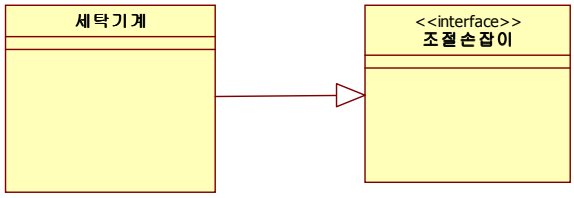
2.5.복합연관

복합체(Composite)는 강한 집합연관에 의해 만들어진 클래스이다. 복합체에서 각 컴포너트 클래스는 오직 하나의 전체 클래스에만 소할 수 있다. 커피 테이블이 복합체의 예인데, 몸체와 다리로 구성되어 있다. 복합연관의 표기는 집합연관을 나타내는 방법과 동일하며, 단지 마름모에 색깔만 칠하면 된다.

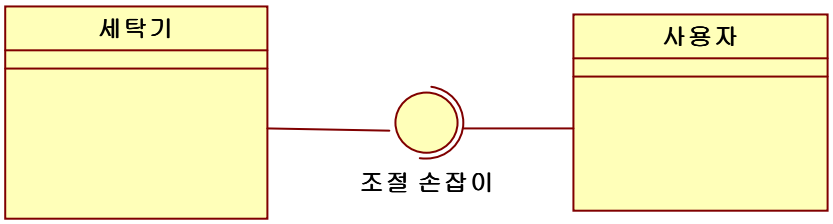


2.6.인터페이스와 실체화

인터페이스(Interface) 는 클래스의 일정한 행동(behavior)을 나타내는 오퍼레이션의 집합으로써, 다른 클래스에서 사용될 수 있다.
UML에서는 이런것을 실체화(Realization)관계 라고 한다.
인터페이스의 모델링 방법은 클래스의 모델링 방법과 비슷하며, 차이가 있다면, 인터페이스는 오퍼레이션의 집합이기 때문에 속성(attribute)를 갖지 않는다. 그리고 인터페이스의 이름 위에 <<interface>>라고 써주면 해결된다.

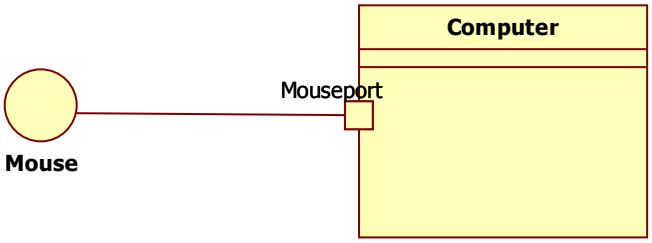


그리고 클래스와 인터페이스 사이의 실체화 관계는 다음과 같이 간단하게 나타낼 수 있다.



2.6.1.인터페이스와 포트

UML2.0에서는 클래스와 인터페이스의 연결단자를 그릴 수 있어서 인터페이스의 개념을 좀 더 자세히 표현 할 수 있다.



그것을 Port라고 하며 인터페이스와 클래스의 연결단자를 좀더 자세하게 나타낼 수 있다.

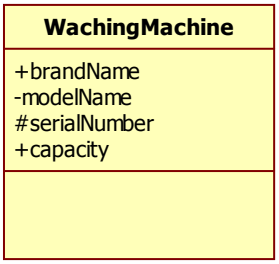
2.7.가시성(Visibility)
가시성은 속성과 오퍼레이션에 적용되는 것으로, 해당 클래스의 속성과 오퍼레이션을 들여다 볼 수 있는 범위를 말한다. 가시성을 나타내는 세가지 protected, public, private의 개념을 들어보면 알기 쉽다.

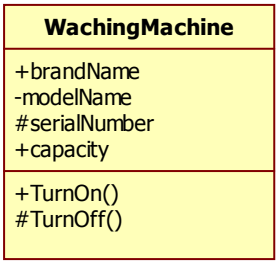
Public 속성과 오퍼레이션은 다른 클래스가 마음껏 사용 할 수 있다.
Protected속성과 어퍼레이션은 원래 클래스와 여기서 상속받은 클래스만이 사용할 수 있다.
Private속성과 오퍼레이션은 원래의 클래스만 사용할 수 있다.

인터페이스를 실체화 하기 위해서는 인터페이스 안에 설정된 오퍼레이션들이 모두 public 가시성을 가지고 있어야 한다. protected나 private로 막힌 오퍼레이션은 아무런 의미가 없다. 왜냐면 인터페이스는 많은 클래스들에의해 실체화되기 위한 용도로 고안되었기 때문이다.

UML에서 속성이나 오퍼레이션의 가시성을 설정하는 기호는 +,#,-로서 속성이나 오퍼레이션 이름 앞에 붙여준다.

위와 같이 해당하는 속성이름 앞에 public, private, protected가 표시되어있다.
오퍼레이션들도 마찬가지로 같은 표시법을 쓴다.





2.8.스코프

속성과 오퍼레이션이 가지고 있는 또 하나의 특성으로써 스코프(scope)가 있다. 스코프에는 두가지 종류가 있다. 인스턴스 스코프(instance scope)에서는 각각의 인스턴스에 속한 속성과 오퍼레이션들이 각자의 값을 가지도록 되어있다. 클래스 스코프(classifier cope)에서는 해당 클래스에 대해 유일한 속성값과 오퍼레이션 값을 가진다. 클래스 스코프가 설정된 속성과 오퍼레이션은 이름에 밑줄이 그어져 있다. 클래스 스코핑은 특정한 그룹의 인스턴스들이 그 클래스에 고유한 속성을 공유해야 할 필요가 있을 때 사용한다. 인스턴스 스코핑은 일반적으로 사용되는 스코핑이다.

유스케이스 (USE-CASE)

1. Use Case Diagram 정의

Use Case Diagram은 UML을 이용한 새로운 시스템 설계의 시발점이 된다. 시스템에 연루된 모든 사람들이 이해할 수 있는 방법으로 시스템 비즈니스 요구사항들을 열거하는데 이용된다.
Use Case Diagram은 시스템 세부사항에 대한 최상위 수준의 형태화 모형이며, 관리자, 고객들, 그리고 다른 비개발자들 사이에 개발 완료 후 시스템 작동내역에 대하여 원활한 의사소통 방법을 제공한다. Use Case Diagram은 시스템 작동에 관한 세부사항은 보여주지 않으며, 누가 시스템을 사용하는지와 시스템을 통하여 할 수 있는 일이 무엇인지를 묘사한다.

2. Use Case Diagram 작성 시기
- 소프트웨어 프로젝트의 개발범위를 정의하는 단계

- 소프트웨어에 대한 요구사항을 정의하는 단계
- 소프트웨어의 세부기능을 분석하는 단계
- 소프트웨어가 아닌 업무영역을 이해하고 분석하는 단계

3. Use Case Diagram 구성요소
- 요소 : Actor, Usecase
- 관계 Communication, Include, Extend, Generalization

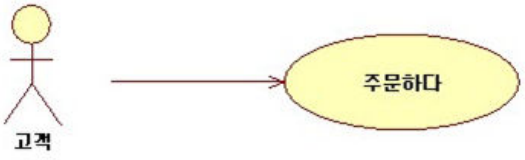
1) Actor

- 시스템이 외부에 존재하면서 시스템과 교류 혹은 상호작용 하는 것
- 시스템이 서비스를 해주기를 요청하는 존재
- 시스템에게 정보를 제공하는 대상

2) Usecase

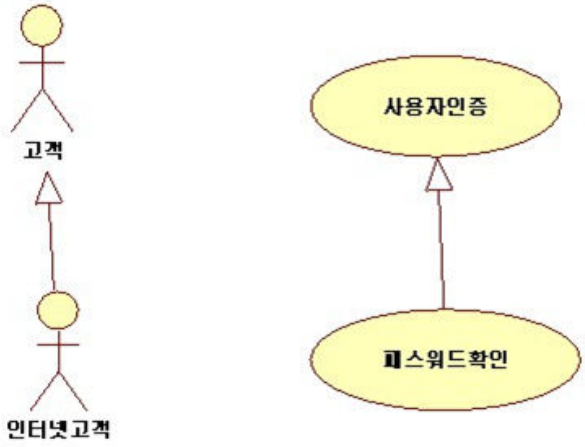
- 시스템이 제공하는 서비스 혹은 기능
- 시스템이 Actor에게 제공하는 사용자 관점의 기능단뒤
- Actor의 요청에 반응하여 원하는 처리를 수행하거나 정보를 제공
- Actor와 한 번 이상의 상호작용을 통한 의미있는 묶음의 시스템 행위
- 의미있는 자기완결형의 서비스 단위
- 사용자관점에서의 정의가 필요

3) Communication



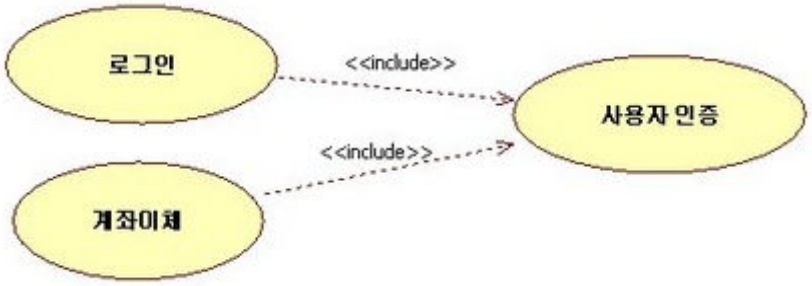
- Actor와 Usecase 사이에 정의되는 관계
- 일반 상호작용 관계가 존재하는 것을 의미
- Actor는 정보를 통보받거나 요구
- Usecase는 정보를 제공

4) Generalization
- Actor와 Actor, Usecase와 Usecase 사이에 정의
- 두 개체가 일반화 관계에 있음을 의미



- 보다 보편적인 것과 보다 구체적인 것 사이의 관계 (is-a 관계)
- 상속의 특성을 지님

5) Include
- Usecase와 Usecase 사이에 정의되는 관계
- 한 Usecase가 다른 Usecase의 서비스 수행을 요청하는 관계
- 즉, 한 Usecase가 자신의 서비스 수행 도중에 다른 Usecase의 서비스 사용이 필요할 때 정의(서비스는 반드시 사용이 되어야 함)
- 포함되는 Usecase는 공통 서비스를 가진 존재



6) Extend

- Usecase와Usecase 사이에 정의되는 관계
- Include와 동일하게서비스 수행을 요청하는 관계
- Include와 달리 서비스가 수행되지 않을 경우도 있음
- 수행 요청 조건을 Extention Point라고 함

4. Use Case Diagram 작성 단계

1 Actor 식별
모든 사용자 역할 식별
상호작용하는 타 시스템 식별
정보를 주고받는 하드웨어 및 지능형 장치 식별

2 Usecase 식별
Actor가 요구하는 서비스 식별
Actor가 요구하는 정보 식별
Actor가 시스템과 상호작용하는 행위를 식별

3 관계정의
Actor와Actor 관계분석 정의
Actor와 Usecase 관계분석 정의
Usecase와 Usecase 관계분석 정의

4 Usecase 구조화
두 개이상의 Usecase에 존재하는공통서비스
추출된 서비스를 Usecase 정의
조건에 따른 서비스 수행부분 분석하여 추출
추출된 서비스를 Usecase로 정의
추출된 서비스를 사용하는 usecase와 관계 정의

협력 다이어그램
(Collaboration Diagram)

1. Collaboration Diagram 정의

Collaboration Diagram은 Class Diagram과 Sequence Diagram의 교차적 개념으로 생각할 수 있다. Collaboration Diagram은 객체, 역할, 이들 간의 순차적 의사소통을 모형화한다.
- 하나의 Usecase를 실현하기 위해서 어떤 객체가 참여하고, 객체 간에 어떤 순서로 상호 협력해야 하는지를 정의한 모델
- 해결해야 할 문제가 주어진 상황에서 그 문제를 해결하기 위해 필요한 객체를 정의하고, 객체간의 동적인 상호관계를 순서에 따라 정의함으로써 주어진 문제를 해결하는 모델
- 모델링하는 공간이 자유롭기 때문에 같은 유형의 객체를 모아 놓을 수 있고, 구조적인 측면을 중시하여 모델링

2. Collaboration Diagram 작성목적

- 객체간 동적 상호작용을 구조적 측면을 중시하여 모델링
  · 어떤 객체가 참여하고 어떻게 상호작용 하는지를 정의하고, 객체간 상호작용을 구조적인 관점에서 정의
- 객체간 상호작용을 정의하는 과정에서 객체를 더욱 상세하게 정의
  · 객체간 상호작용을 정밀하게 정의함으로써 객체간에 분담해야 하는 책임이 더욱 상세하게 정의
  · 객체의 책임은 오퍼레이션으로 구현
  · 행위를 위해 필요한 객체의 속성도 정의
- Usecase 실현
  · Usecase별로 Collaboration Diagram이 작성
  · Usecase에 필요한 객체가 주인공으로 등장하고, 객체간의 메시지를 통해서 Usecase의 기능이 실현
- 프로그래밍 사양 정의
  · 상세화된 Collaboration Diagram은 곧바로 프로그래밍 될 수 있는 수준으로 프로그램 사양 정의

3. Collaboration Diagram 작성시기

- Usecase Diagram이 정의부터 프로그램 코딩을 하기 전 (Sequence Diagram과 동일)
- 다른 Diagram처럼 여러 번에 걸쳐 정제되는 과정
- 단계별 다른 관점의 객체 등장
  · 분석 단계 : 비즈니스 관점
  · 설계 단계 : 구현 관점
- 작성하기 위한 준비물 및 선행과정
  · 유스케이스 다이어그램이 작성

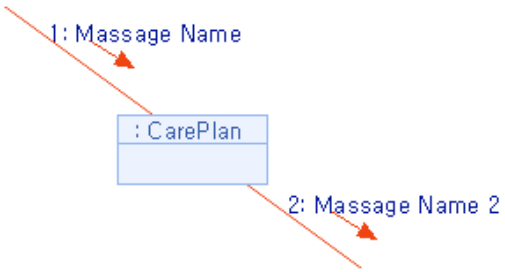
4. Collaboration Diagram 구성요소

① Things
- Actor
  · Sequence Diagram과 동일
- Object
  · Sequence Diagram과 동일

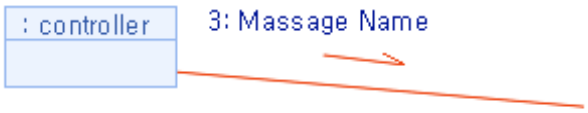
② Relationship
- Message

· Flat Flow of Control
  * 모형에서 알려지지 않거나 중요하지 않은 메시지 유형임을 표시할 경우에 사용

· Nested Flow of Control
  * 모형에서 알려지지 않거나 중요하지 않은 메시지 유형임을 표시할 경우에 사용



· Asynchronous Flow of Control
  * 병렬처리로 의사소통을 할 경우에 사용

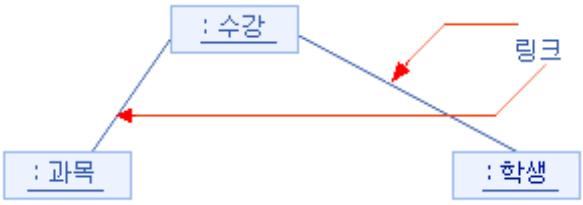


· Return Flow

- Link

· 객체와 객체간 연관관계.

* 메시지는 링크를 따라 움직이므로 객체가 통신하려면 링크되어 있어야 함.



5. Collaboration Diagram 작성순서

① 작성 대상 선정한다.
② Actor를 Diagram에 위치시킨다.
③ Class를 Diagram에 위치시킨다.
④ 객체간 메시지를 정의한다.
⑤ 객체를 추가로 정의한다.

시퀀스 다이어그램
(Sequence Diagram)

1. Sequence Diagram 정의

Sequence Diagram은 시간의 순서화된 순차를 이용해서 객체간의 교류를 모형화는데 이용되며, Usecase 행위를 클래스에 분배하는데 이용된다. Sequence Diagram은 통합 프로세스에서 기술된 소프트웨어 개발 Life Cycle 전반에 걸쳐서 이용된다.

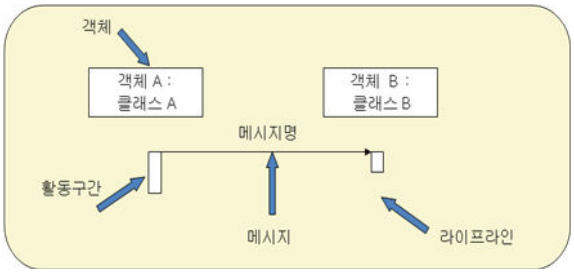
2. Sequence Diagram 작성 목적

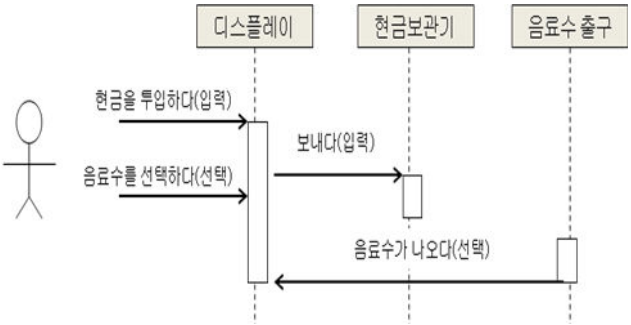
- 객체간 동적 상호작용을 시간적 개념을 중시하여 모델링
- 객체의 오퍼레이션과 속성을 상세히 정의
- Usecase를 실현
- 프로그래밍 사양 정의

3. Sequence Diagram 작성 시기

Usecase Diagram 정의 후부터 프로그램 코딩 전

4. Sequence Diagram 구성요소





① Things

- Actor
  · Usecase 에서의 actor.
- Object
  · 클래스의 인스턴스. 클래스 타입으로 선언된 변수형태로 존재

② Relationship

- Message

· 메시지(Message)는 Sequence Diagram의 서로 다른 객체간의 의사소통을 묘사하는데 이용된다. 객체가 다른 객체의 처리를 유도할 필요가 있을 때나 다른 객체에게 정보를 전달 하고 싶을 때 메시지가 사용됨.

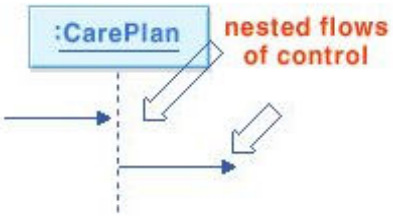
메시지는 호출 활성화 객체의 생명선으로부터 피호출 객체의 생명선까지 화살표로 그려진다. 화살표 위에는 보내지는 메시지가 위치함.

· Flat Flow of Control

* 가장 일반적 메시지

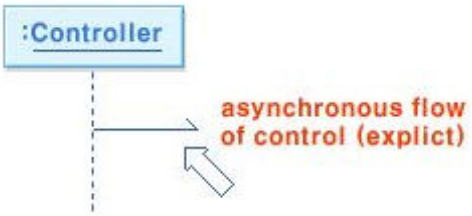
· Nested Flow of Control

* 메시지가 중첩 시 메시지가 모두 돌아와야 다음 처리 진행



· Asynchronous Flow of control

* 메시지의 결과를 기다리지 않고 다음 처리 진행



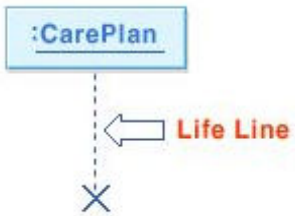
· Return Flow

* 메시지를 처리한 결과. 필요한 경우에만 사용.

③ etc

- Life Line

· 객체의 생존기간. 점선에 X표시가 객체가 소멸하는 시점



- Activation

· 객체가 활성화 되어있는 기간. (점선표기)

· 객체가 외부 메시지를 받고 보낸 메시지를 기다리는 기간. (좁고 긴 사각형)

5. Sequence Diagram 작성 순서

① 작성대상 선정 : Usecase를 선정하고 Usecase 정의서 분석
② Actor 위치시킴 : Actor는 좌측부터 위치
③ 클래스 위치시킴 : Usecase에 참여하는 클래스 위치
④ 객체간 메시지 정의 : 시간순서대로 객체간 메시지 정의

⑤ 객체 추가정의 : 요구사항 처리를 위해 필요한 객체가 정의되지 않았으면 추가 정의.

활동 다이어그램 (Activity Diagram)

1. Activity Diagram 정의

Activity Diagram은 시스템 분석시 Usecase Diagram의 다음 단계로써 시스템에 필요한 여러 표기법들을 제공한다. Activity Diagram은 사용자에게 시스템 실행의 예측과 여러 조건과 자극에 대하여 시스템이 어떻게 반응하는지를 보여준다. Activity Diagram은 모형화 단계의 시스템 설계시 복잡한 객체 행위를 모형화하는데 이용될 수 있다.

2. Activity Diagram 작성 목적

- 처리순서 표현
- 비즈니스 프로세스 정의
- 프로그램 로직 정의 : 처리흐름의 도식화로 프로그램 로직 정의 가능
- Usecase 실현

3. Activity Diagram 작성시기

- 업무 프로세스 정의 시점
  · 비즈니스프로세스를 정의하는 용도로 Activity Diagram을 작성 가능
- Usecase 정의서 작성 시, 처리절차 기술할 때
- 오퍼레이션 사양 정의시
  · Class 오퍼레이션의 사양을 Activity Diagram을 적용하여 작성가능
- 기타
  · 기타 처리흐름이나 처리절차가 필요한 시점이면 언제나 Activity Diagram 작성 가능

4. Activity Diagram 구성요소

① Things

- Activity

· 행위나 작업 (내부적으로 구조를 가지는 단위)

· Activiy의 크기는 작성 대상에 따라 유동적이며, 한 Activity Diagram에서는 Activity의 크기 균일한 것이 바람직

· Activity는 최소 단위가 아니며 내부적으로 구조를 가질 수 있는 단위

· Activity는 해당 작업의 종료 시점을 명확히 정의하기가 힘듬 ·

· ex) 상품조회, 구매결정, 결재내용입력, 결재자지정

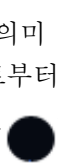
■ 표기법 : Acitivity는 모서리가 둥근 사각형으로 표기하며, 이름은 Symbol 내에 표기

- Initial State

· 처리 흐름이 시작하는 곳을 의미

· 모든 처리 흐름은 시작점으로부터 개시되어 전개

■ 표기법 : 속이 꽉 채워진 원

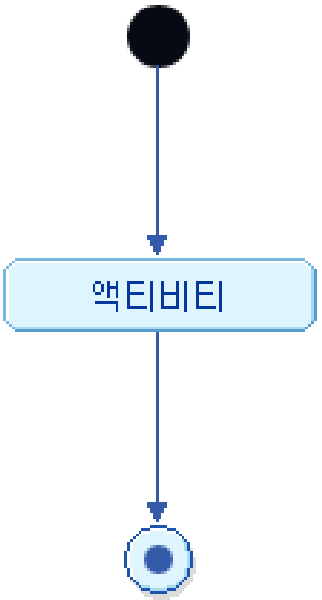


- Final State :

· 처리 흐름이 종료하는 곳을 의미

· 모든 처리 흐름은 종료점에서 처리 흐름을 완료

■ 표기법 : 속이 채워진 원에 바깥의 또 다른 원이 둘러싸고 있는 모양



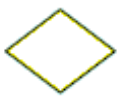
- Decision(Branch)

· Decision은 분기가 일어나는 곳

· 논리식의 결과 값에 따라 두 곳 이상의 흐름으로 분기가 일어남

· 처리 흐름은 논리식의 결과에 따라 처리 흐름이 나누어져 전개되는 것은 매우 흔하게 일어나는 일

■ 표기법 : 속이 빈 마름모꼴로 표기하며, 명칭이나 기타 장식이 붙지 않음



- Synchronization Bar

· 병렬 처리절차가 시작되거나 모이는곳

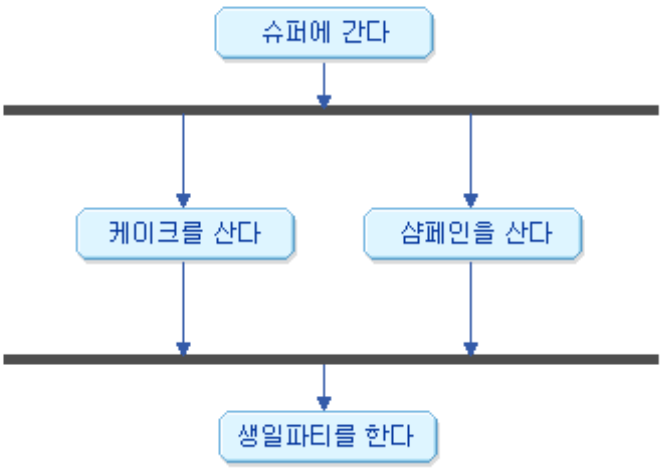
· 종종 둘이상의 처리절차가 그 수행순서에 상관없이 병렬로 진행될 경우 있음

· Synchronization Bar로부터 분기해서 다음 Synchronization Bar로 모일 때 까지의 처리 절차는 병렬로 수행

· Synchronization Bar에 이어진 Activity가 수행되기 위해서는 병렬로 수행되는 Synchronization Bar상의 모든 처리절차가 끝나야함

■ 표기법 : 두꺼운 실선으로 표기, 대부분 수평선으로 표기되나, 수직선으로 표기할 수도 있음

ex)



② Relationship

- Transition

· 하나의 Activity가 행위를 완료하고 다른 Activity로 처리순서가 옮겨지는 제어흐름 표현

· 하나의 Activity에서 여러 개의 Transition이 나가기도 하고, 들어오기도 함.

■ 표기법 : 화살표가 달린 실선으로 표기하며, Activity의 배치에 따라 수평선이나 수직선으로 표기

③ Swim lane

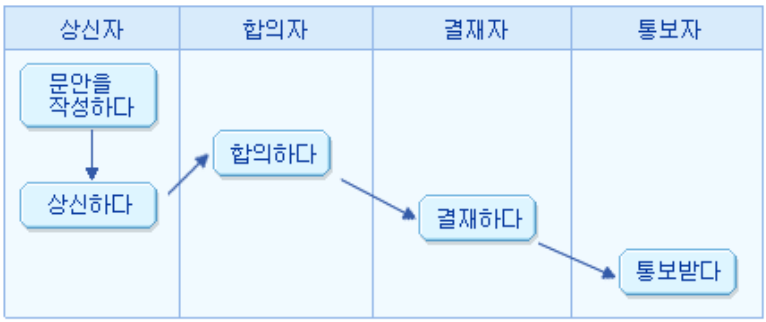
· Swim lane은 여러 가지 용도로 쓰일 수 있음

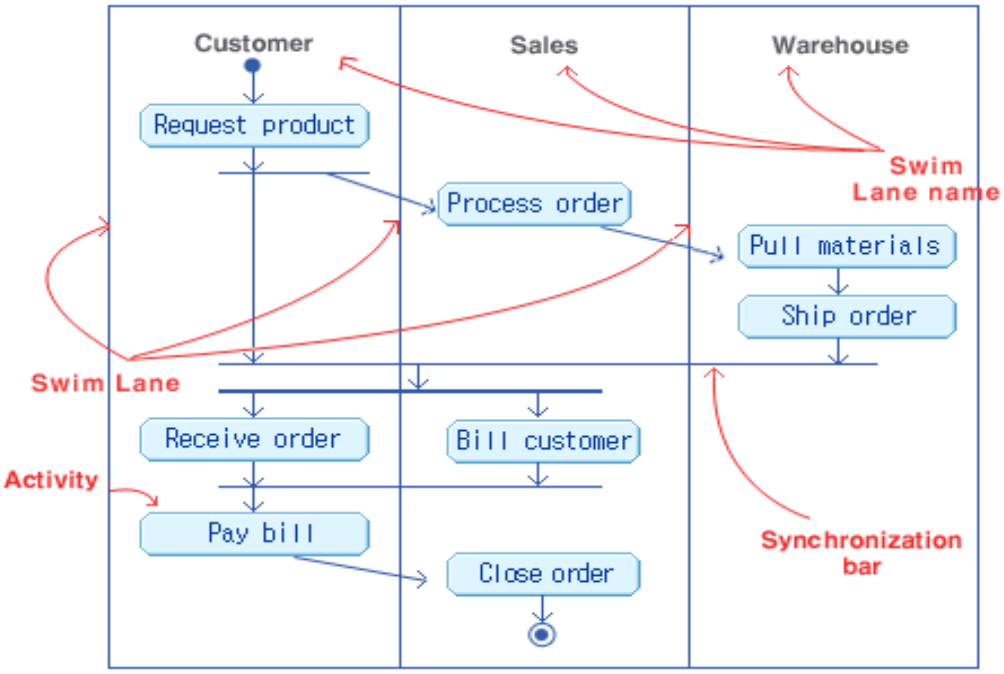
* 업무 조직의 구분일 수 있고, 개인의 역할에 따른 구분

· Swim lane의 영역 내에 정의된 Activity는 그 Swim lane이 관장하고 Ownership을 가짐

· Swim lane을 표현함으로써 누가(Swim lane) 무엇을 한다(Activity)라는 식의 표현이 가능

■ 표기법 : Swim lane은 영역으로 표현을 하며, Activity Diagram의 제일 위쪽에서 아래쪽까지 수직방향으로 공간을 구분하는 방식으로 표현. 모양이 마치 실내 수영장의 트랙 같다고 해서 명명.





5. Activity Diagram 작성순서

① 작성대상 선정 : 업무프로세스 모델링, 오퍼레이션 사양 정의

② Swim lane 정의 : 대상영역에 명확한 역할을 정의해야 할 때.

③ 처리절차 모델링 : 시작점, 끝점 반드시 표현.

상태 다이어그램 (Statechart Diagram)

1. Statechart Diagram 정의

Statechart Diagram은 객체가 상태를 변화시키는 방법을 모형화하는데 쓰인다. 상태는 특정시점에서 객체 행위의 이정표가 되거나 객체의 단적인 표현으로 정의되기도 한다.

2. Statechart Diagram 작성목적

- 객체의 상태변화를 상세히 분석
- event에 의한 객체의 반응을 분석
- 객체의 속성이나 오퍼레이션을 검증

3. Statechart Diagram 작성시기

한정해서 정하기 어려우나, 보통 Class가 정의된 후, 즉 Class Diagram과 Sequence Diagram이 작성된 후에 작성.

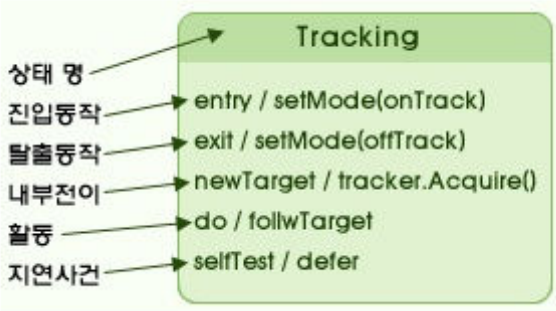
4. Statechart Diagram 구성요소

○ 시작상태(Initial state), 종료상태(Final State)

	표기	의미
시작 상태	 시각점은 속이 꽉 채워진 원으로 표기	시작 상태는 객체의 상태변화가 시작되는 곳을 의미합니다. 보통 객체의 생성시점이 시작상태가 됩니다.
종료 상태	 속이 채워진 원에 바깥의 또 다른 원이 둘러싸고 있는 모양으로 표기	종료 상태는 객체 상태변화가 종료하는 곳을 의미합니다. 보통 객체의 소멸시점이 종료상태가 됩니다.

- State
  · State란 객체가 가질 수 있는 조건이나 상황
  · 생명주기 동안 객체의 State는 변화하며, State는 객체의 특정한 속성의 값으로 표현
  · 예) 자동차 객체의 State : “주차”, “주행”, “정차”, “수리”

■ 표기법

기본형 표기	상세형 표기
- 모서리 둥근 사각형으로 표기 - 상태 명은 심볼 내에 표기	- 수평으로 구분된 모서리 둥근 사각형으로 표기 - 상태 명은 위쪽부분에, 동작과 그 외의 부분은 아랫 부분에 표기
	

· 진입 동작(Entry Action)
상태에 들어올 때 수행되는 동작을 정의

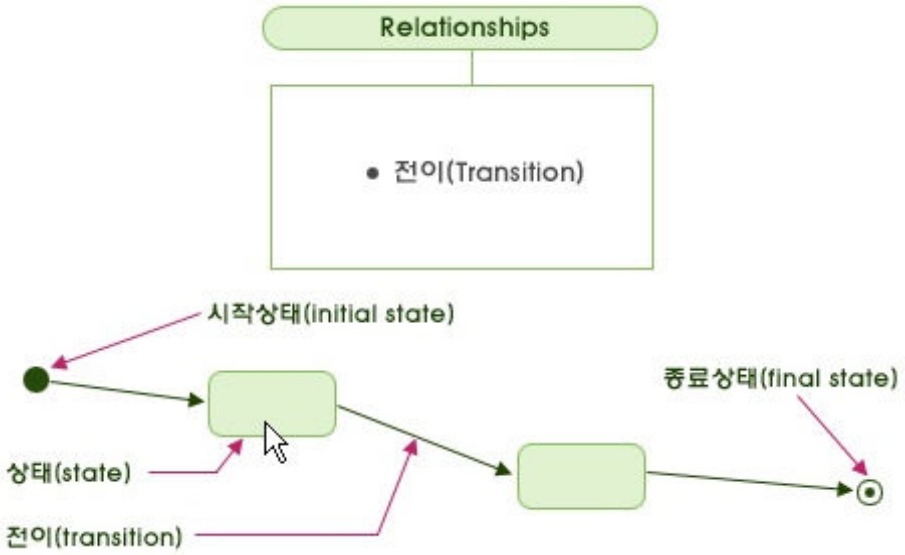
· 탈출 동작(Exit Action)
상태에 나갈 때 수행되는 동작을 표기

· 내부전이(Internal Transition)

현재 상태에서 처리할 수 있는 이벤트가 발생할 경우 상태를 떠나지 않고 해당 사건을 처리하는 경우

· 활동(Activity/Action)
현 상태에서 수행할 동작을 표현

· 지연 사건 (Deferred event)
현 상태를 빠져 나갈 때 발생한 것처럼 그 효과를 지연시킨 이벤트. 위 예에서 Tracking 상태에서 selfTest 이벤트가 발생하면 이것을 메시지 큐에 저장 했다가, Tracking 상태에서 벗어나는 순간 이벤트가 활성화.

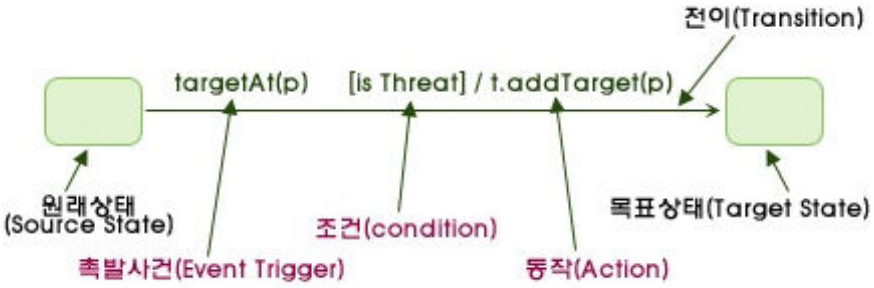


- Transition
· Transition이란 하나의 상태에서 다른 상태로 변화하는 것이며 상태 간의 관계를 의미.

■ 표기법
* 전이는 상태와 상태 사이에 화살표가 달린 실선으로 표기.
* 선 위에는 촉발사건(Event Trigger), 조건(Condition), 동작(Action)이 차례로 표기.
* 위 세 가지 표기내용은 각각 생략될 수 있음.

· 원래 상태 (Source State)
* 전이가 실행되기 전의 객체 상태

· 촉발 사건 (Event Trigger)
* 전이를 촉발시키는 사건



· 전이 조건(Condition)
 * 전이 촉발 시에 검토되는 Boolean 식 (참일 경우에만 전이가 수행됨)

· 동작(Action)
 * 전이 도중 실행되는 행위 또는 오퍼레이션

· 목표 상태(Target State)
 * 전이가 완료된 후의 객체 상태

5. Statechart Diagram 작성순서

① 좀더 상세히 다루어야 할 개체(객체나 Usecase)를 식별한다.
② 각 개체의 시작 상태와 종료 상태를 식별한다.
③ 각 개체와 연관하는 사건을 결정한다.
④ 시작 사건으로 시작하는 상태차트 다이어그램을 생성한다.
⑤ 필요한 곳에서 복합상태를 생성한다.

컴포넌트 다이어그램 (Component Diagram)

1. Component Diagram 정의

- 시스템의 구현관점에서 실행모듈(Component)을 정의 하고, 실행모듈간의 정적 상호작용을 정의 한 모델
- 시스템이 어떠한 물리적 구성요소들로 -실행모듈(Component)- 구성되고, 그들간의 연관성을 정의
- SW 분야에서 사용되는 컴포넌트 정의
 1) 컴포넌트는 시스템의 재사용 가능한 구성요소
 2) 컴포넌트는 시스템의 교체단위이자 업그레인 단위
 3) 컴포넌트는 인터페이스를 통해 그 기능이 사용되어지는, 독립적으로 인도되는 기능 조각

2. Component Diagram 작성목적

- 시스템의 실행모듈(Component)들을 정의
· 시스템이 구축될 때, 어떤 Component들로 구축 될 것인지를 정의하는 용도로 사용
· 컴포넌트는 독립적으로 빛, 교체가 가능한 단위
· 개발 플랫폼에 따라 이러한 Component의 특성은 달라짐

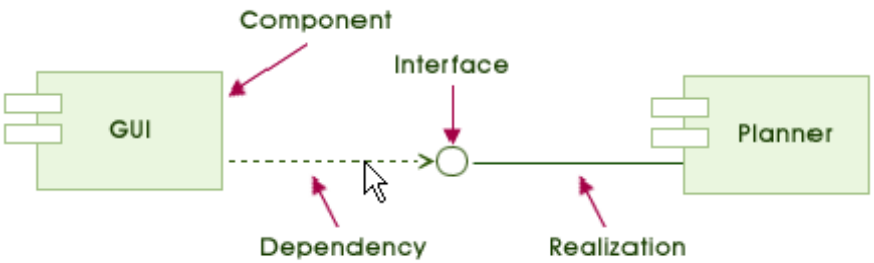
- Component간 Dependency를 정의
· Component간의 정적인 상호작용을 정의하는 용도로 사용
· Component 사이의 종속관계를 표현함으로써 실행시 상호참조하는 연관성을 표현

- Component뿐만 아니라 소스코드, 데이터베이스 등의 상호작용을 모델링

3. Component Diagram 작성시기

- Component Diagram은 시스템의 설계단계의 막바지에 작성.
- 모든 Class가 물리적으로 완전히 정의되고, 그 상호관계도 정의된 후 Component Diagram 작성.

4. Component Diagram 구성요소



① Things

- Component

· Component는 독립적으로 배포되고 교체되며 재사용될 수 있는 SW조각 의미

■ 표기법 : Component는 탭이 달린 직사각형으로 표기하며, Component 이름은 Symbol 내에 표기

- Interface

· Class의 일종

· Class나 Component의 기능을 외부에 공개할 목적으로 쓰이며, 구현은 하지 않음

· Interface의 구현은 Class나 Component에서 하게 되며, 이 Class는 Interface를 상속하여 단지 선언뿐인 Interface의 구현을 담당

· 단독으로 표시되는 경우는 거의 없으며 해당 Interface를 구현하는 Class나 Component에 붙어 다님

■ 표기법 : Interface는 두 가지 형태로 표기

1) Icon형태의 표기로 원으로 표현, Interface 이름은 아래쪽에 표기.

2) 보통 클래스에 <>라는 stereo type이 부가된 표기

② Relationship

- Dependency

[icon 형태로 표현할 경우] [스테레오 타입으로표현할 경우]

· 객체나 Component가 다른 객체나 Component의 실행을 요청하는 경우, 사물 간의 실행 혹은 참조관계를 표현

· 사용되는 관계

* Class와 Class
* Package와 Package
* Component와 Component
* 때로는 Class-Package-Component 상호 간

■ 표기법 : 점선 화살표로 표현하고, 필요에 따라 선 위에 설명을 붙이기도 함

- Realization

· 정의하는 사물과 이를 구현하는 사물 간에 표현하는 관계

· Realization은 사이에 허용되는 관계

* Interface(정의) - Component(구현)
* Usecase(정의) - Collaboration(구현)
* Interface(정의) - Class(구현)

· 삼각형이 붙은 쪽이 정의하는 사물, 반대쪽이 구현하는 사물

■ 표기법 : 속이 빈 삼각형의 화살표가 한쪽에 달린 점선으로 표현, 특별히 Component Diagram에서는 Interface와 Component간의 실선 표현

Realization의 표기	Interface와 Component 간 표기
	

5. Component Diagram 작성순서

① Component 대상을 정의
· Component Diagram을 그리기 전에 정의 필요
  * 무엇을 Component로 표현, Class를 구성요소로 하는 실행모듈
  * 소스코드를 정의, 기타 무엇을 Component로 표현
② Component를 식별
· Component Diagram에 등장할 Component를 정함.
  * 소스 파일일 경우 그 대상은 쉽게 식별되지만 실행모듈일 경우 간단치 않음
  * 여러 가지 가능한 방법으로 Component를 식별해 내는 작업을 수행
③ Component를 배치하고 필요시 인터페이스를 붙임
· Component Diagram에 Component를 배치하고 이름을 정의
· Interface를 정의할 필요가 있을 경우 Interface를 정의하고 Component와 Realization관계로 연결
④ Dependency를 정의
· Component와 Component간 의존관계를 분석하여 Dependency 관계를 정의

배포 다이어그램 (Deployment Diagram)

1. Deployment Diagram 정의

Deployment Diagram은 각 소프트웨어 구성요소들이 하드웨어의 어디에 위치하는지와 하드웨어 구성요소들이 서로 어떻게 교류하는지를 보여주는 Diagram.

2. Deployment Diagram 작성목적

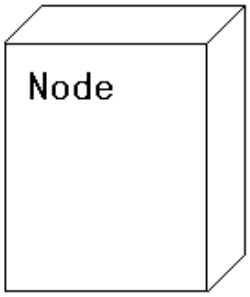
- 각 소프트웨어 구성요소들이 하드웨어의 어디에 위치하는지와 하드웨어 구성요소들이 서로 어떻게 교류하는지를 기록하는데 이용된다.

3. Deployment Diagram 작성시기

- 시스템 설계의 마지막 단계에서 작성
· 모든 설계가 거의 마무리되어 시스템의 하드웨어 사양도 확정된 후 작성

4. Deployment Diagram 구성요소

① Things
- Node
· Node는 하드웨어 구성요소를 표현하는데 이용
■ 표기법 : Node의 이름을 포함하는 3차원 박스로 표기



② Relationship
- Communication association
· Node들은 Communication association으로 연관하며, 이 관계는 두 가지 하드웨어 구성요소(Node)가 나름대로의 의미를 가지고 서로 통신하고 있음을 알게 해주며, 통신 연관관계와 함께 항상 표시되는 고정유형에 나타난다.
■ 표기법 : · 굵은 선으로 한 노드에서 다른 노드로 그려진다.
· 통신 연관관계의 고정유형은 Component 종속성을 위한 고정유형처럼 (<<...>>)로 표기된다. 고정유형 이름은 통신수단을 표시하거나, 하드웨어 두 구성요소들 사이의 프로토콜을 묘사하는 대상이 된다.

5. Deployment Diagram 작성순서

① 시스템의 노드를 식별
· 식별 가능한 모든 하드웨어 목록과 일치하는 요구사항들을 조사해야 한다.
② Communication association을 추가
· Communication association으로 식별된 노드들을 연관시켜야 한다.
· Communication association은 Node간의 통신 유형을 표시하기 위해서 고정유형화되기 때문에, 이러한 정보를 위한 요구사항 검색이 필요하다.