Introduction to OOAD
using UML tools

(instructor :: 유준범 교수님)

Konkuk univ. dept of CSE

Team 4
200611517 정훈섭
200711420 권준수
200711448 오희수
200710118 유희찬

Object Oriented Analysis & Design

Contents

What is OOAD??

OOAD

Object-**O**riented **A**nalysis and **D**esign(OOAD)는 상호 작용하는 **객체들 간의 그룹으로 시스템을 모델링**하는 software engineering입니다. 각각의 객체는 시스템에서 한 entity를 나타내고, 객체가 가지고 있는 상태와 동작은 '**class**' 로 묶여있습니다. 이 객체들의 조합으로 다양한 시스템 모델을 구현할 수 있습니다. 이런 모델들을 표현하는데 여러 가지 방법이 쓰이는데 UML(Unified Modeling Language)이 대표적인 방법입니다.

OOA

Object-**O**riented **A**nalysis 는 시스템의 **기능적 요구사항(functional requirement)을 분석**합니다. OOA는 시스템이 어떤 일을 하는지 - 즉, '**What**' 에 - 초점이 맞쳐져 있습니다. OOA는 domain으로부터 개념적인 모델을 만들기 위한 정보를 분석합니다. OOA 단계에서는 동시성, 지속성과 같은 개발 단계에서의 제한에 전혀 구애를 받지 않습니다. 이들은 다음 단계인 OOD에서만 영향을 끼치죠.

요구 명세서(requirements statement), 이해관계자(stakeholder) 혹은 이해 집단과의 인터뷰 내용 등이 분석에 필요한 자료가 됩니다. 시스템은 (기술적, business적으로) 여러 상이한 domain으로 나눠질 수 있고, 분석 시에 이들 모두 각각 따로 분석을 해야합니다.

OOA를 통해서 기능적인 요구사항(functional requirement)을 만족하는 시스템의 모델을 그리게 되고(그리고 이 모델은 대게 use case나 UML class diagram 등으로 표현됩니다), 이 모델을 통해서 시스템이 고객의 요구사항을 만족하는 시스템을 개발하는데 그 목표가 있습니다.

OOD

Object-**O**riented **D**esign(OOD)는 시스템이 어떻게 - 즉, '**How**' - 동작하는지에 초점이 맞춰져 있습니다. OOA를 통해 만들어진 개념적인 모델을 기술 또는 환경과 같은 **비기능적인 (non-functional) 제한을 걸어 수정**을 합니다.

OOA에서 모델링 됐던 객체들은 OOD에서 class와 interface로 구현이 되고, 결과적으로 OOD를 통해 시스템이 어떻게 작용하는지 명확한 명세서를 얻을 수 있습니다.

What is UML??

객체 관련 표준화기구인 OMG에서 1997년 11월 객체 모델링 기술(OMT;object modeling technique), OOSE 방법론 등을 연합하여 만든 **통합 모델링 언어**로 **Object Oriented Analysis & Design 방법론의 표준 지정을 목표**로 하고 있습니다.

characterstic of UML

모델링에 대한 표현력이 강하고 비교적 모순이 적은 논리적인 표기법(notation)을 가진 언어로,
= 요구분석, 시스템 설계 및 구현 등의 과정에서 생기는 **개발자 간의 의사소통이 쉬워**지고,
= 생략되거나 불일치되는 **모델링 구조에 대한 지적 용이**하며
= 개발하려는 시스템 **규모에 상관없이 모두 적용 가능**합니다.

또한 공정간의 명확하여,
= 다수의 다이어그램이 업무나 시스템을 여러 가지 시점으로 표현이 가능하고,
= 다이어그램 간의 대응관계가 일관성 있고, 명확하게 표현됨으로써
= 상위 작업부터 하위 작업에 이르기까지 명확한 개발 가능해 집니다.

Represent of UML

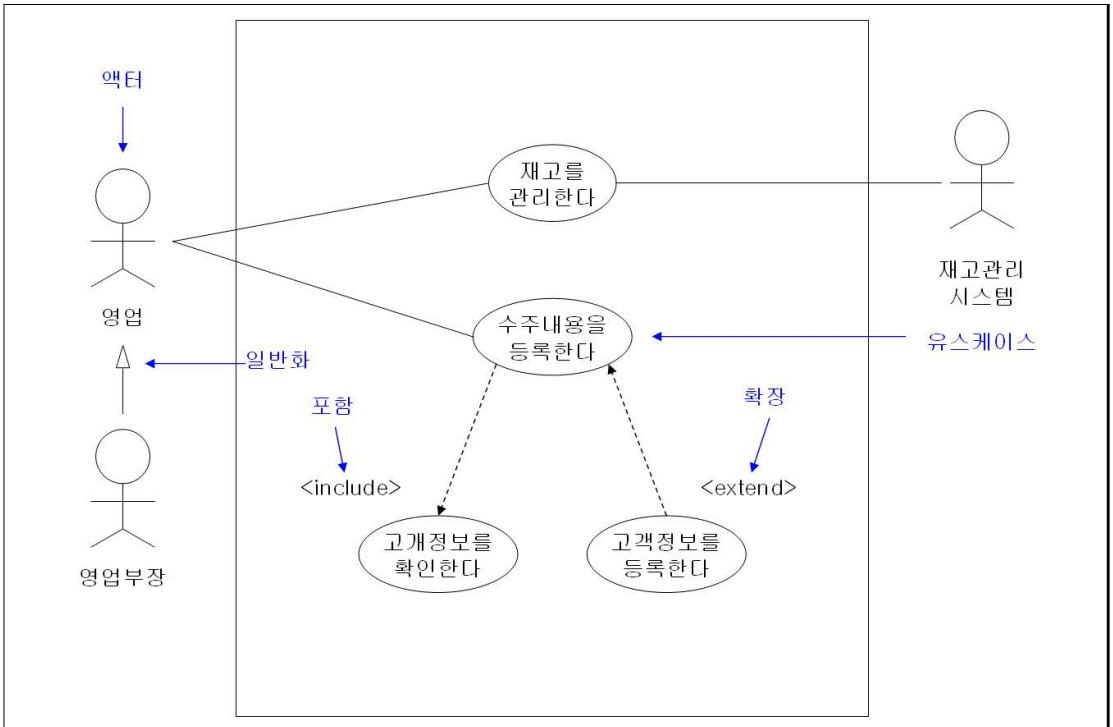
UML을 표현하는데 diagram이 사용되고, 다음과 같은 diagram 종류가 있습니다.

구조성	Class Diagram	모델의 조립 부품의 집합. 클래스의 구조와 클래스간의 관계를 표현
	Object Diagram	시스템의 어떤 시점에서의 스냅샷
행동성	UseCase Diagram	시스템의 문맥과 외부기능(functionality)의 설정
	Sequence Diagram	상호작용하는 객체의 시간 순서 계열 즉, 객체의 집단 메시지 송신에 대한 시계열 표현
	Collaboration Diagram	객체 집단에서의 상호작용에 대한 직접적 표현이나 객체 집단의 접속망의 형태와 메시지, 스레드의 순서 등을 표현
	State Machine Diagram	1개의 객체 생성에서 소멸까지 상태전이. 즉, 어떤 클래스에 속하는 객체의『라이프사이클 표현』을 제공
	Activity Diagram	1개의 interaction 전체에서의 순서 제어 플로우. 상태 다이어그램의 상대적 표현으로서 워크플로우에 초점을 맞춘 Diagram
구현성	Component Diagram	소프트웨어-유닛 간의 의존관계를 나타내는 것으로 소프트웨어 모듈 구성이나 버전 관리도 표현 할 수 있다.
	Deployment Diagram	객체나 패키지, 파일 등을 실제 플랫폼이나 네트워크 노드 상의 어디에 배치할 것인지, 그리고 어느 프로세스 상에서 실행할 것인지라는 물리적인 관점에서 시스템 구성을 표현

이 중 몇 가지 diagram을 보다 자세하게 살펴보도록 하겠습니다.

Use case diagram

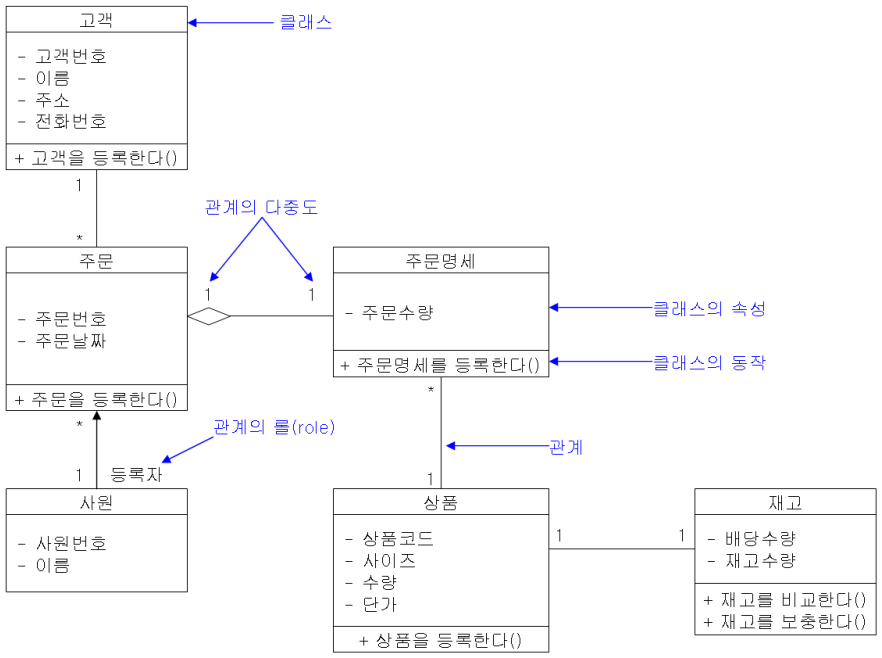
외부로부터 본 시스템의 기능(행동)을 표현하여,
= 사용자 등이 이 시스템에 어떠한 기능이 존재하는가를 확인 할 수 있고
= 요구 분석 등의 상위 작업에서 사용(요구를 정리하여 개발 대상을 명확하게 하기 위해)합니다.



Actor	= 시스템이 서비스를 해주기를 요청하는 존재 = 시스템에게 정보를 제공하는 대상 = 사람모양의 아이콘으로 표기(스틱맨)
generalization	= Actor를 추상화하거나 구체화할 경우에 속이 빈 삼각형이 달린 실선으로 표기 =객체지향의 '상속'과 비슷한 의미
Use case	= 아래에서 다시 설명
include	=어떤 기능이 다른 기능을 포함할 경우 Use case간에 포함 관계를 사용해서 표현
extend	= 어떤 기능을 확장해서 기능(처리)을 추가할 경우에 Use case간에 확장의 관계를 사용해서 표현

Class diagram

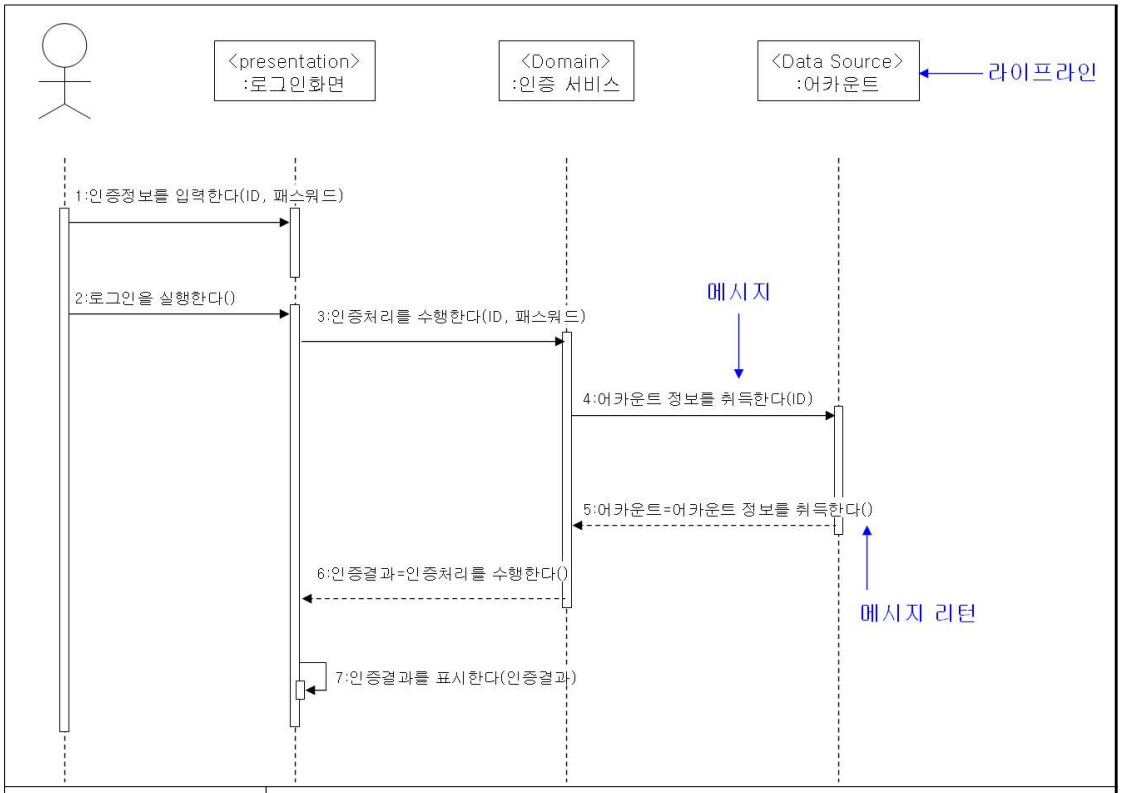
클래스의 구조를 표현하는 다이어그램으로,
= 상위단계 작업에서는 시스템화 대상의 개념적인 구조를 정리하기 위해 사용하고,
= 하위단계 작업에서는 시스템의 행동을 고려, 클래스의 구조를 설계하여 시스템의 재사용성이나 사양 변경에 의한 대응을 쉽게 합니다.



Class	= 객체를 추상화한 형태로 표현 = 클래스명, 클래스의 특징(속성), 클래스의 동작(메소드)의 3요소로 구성 = 속성이나 메소드에는 타입이나 가시화(접근제어자)를 정의할 수 있다.
Relationship	= Class간에 무슨 어떠한 관계가 있는지 나타내는 것으로 실선을 그어 표기 = 다중도(multiplicities)나 관계에 있어서 Class가 수행할 역할을 롤(role)로서 정의 = Class간 관계가 있는경우, instance간에 메시지의 교환이 발생하는 것을 나타낸다. = 메시지 교환에 방향성이 있을 경우 유도 가능성 정의, 화살표를 붙여서 표현 = Class간에 전체와 부분의 관계가 있을 경우(집약관계), 마름모를 끝에 붙여서 표기
Generalization	= 어떤 Class를 추상화해서 다른 Class를 정의하는 것 (SuperClass) = 속이 빈 삼각형달린 실선으로 표현
Specialization	= 어떤 Class를 구체화해서 다른 Class를 정의하는 것(SubClass)
Inheritance	= SubClass가 SuperClass의 특성(속성, 메소드, 관계 등)을 모두 받아서 사용하는 것
Dependency	= 어떤 Class가 다른 Class에게 어떤 형태로 의존하고 있는 것을 나타냄 = 어떤 Class가 동작하는 중에 일시적으로 다른 Class를 사용하는 경우 등에 사용 = Class간 연결강도가 강하지 않음. 점선 화살표로 표현

Sequence diagram

상호작용 diagram의 한 종류로,
= 기능 실현을 위한 객체 간 메시지의 교환을 시간 순서대로 표현



Life Line	= 어떤 기능을 실현하기 위해 상호 작용에 추가하는 객체를 나타냄 = 점선은 객체가 생존하는 시간을 위에서부터 아래로 나타내고 x가 덧붙여진 경우 객체의 소멸시점을 나타냄
Message	= Life Line이 가지는 메소드의 호출을 나타내는 것 = 객체간의 통신에 사용 = 각 라이프라인의 점선을 잇는 실선의 화살표로 표기 = 동기메시지 : 메시지가 중첩될 때 수행하는 작업이 완료 할 때까지 다음 메시지의 송신을 기다리는 것 = 비동기메시지 : 수행하는 작업이 완료하는 것을 기다리지 않고 다음 메시지를 송신
Return	= 라이프라인의 동작이 종료하는 타이밍을 표현 = 메시지 반환을 점선 화살표로 표시 = 수행한 메소드에서 반환 값이 있을 경우 기술

What is Use case?

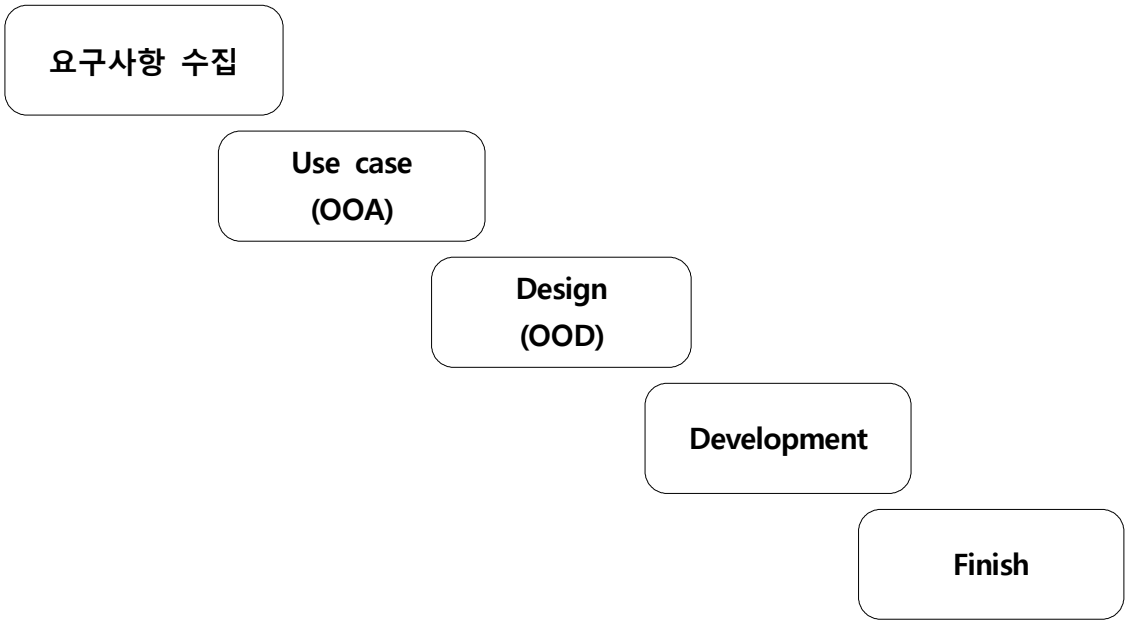
시스템 개발 시 개발될 시스템의 개개 actor에게 측정 가능한 결과를 제공하기 위해 개발될 시스템 안에서 수행되어야 하는 일련의 트랜잭션을 **Use case**라 합니다. **즉, 액터가 시스템 사용 목적을 잘 달성할 수 있도록 개발될 시스템이 제공해야하는 서비스**입니다.

Then, what is Scenario?

액터가 시스템을 사용하면서 벌어질 **여러 가지 상황들을 가정한 일련의 흐름**으로,
= 예외나 에러상황을 접하지 않는 일반적 상황,
= 비정상적인 경우에 발생. 시스템을 정상적인 상태로 되돌릴 수 있도록 대응해야 하는 예외 상황,
= 시스템에 발생해서는 안 되는 것. 일반적으로 에러 상황
등의 actor와 시스템간의 상호 작용에 대한 내용으로 작성됩니다. 또한 서비스 계약에 있어서 해당 역할의 책임은 상대방에 의해 부여됩니다. (actor↔서비스)

Sequence of OOAD development - part I

OOAD 개발 방법은 크게 요구사항 수집, use case 작성(Analysis), 디자인, 개발, 출시 이렇게 5단계로 나눌 수 있습니다. 여기서는 OOAD 개발의 각 단계가 무엇인지, 각 단계에서 어떤 일을 하는지 살펴보도록 하겠습니다.



요구사항 수집

첫 번째로 software 개발에 앞서 engineer는 software가 고객이 원하는 기능을 수행하도록 고객이 원하는 것을 알아내야 합니다. 반대로 말하면 engineer가 개발한 software가 고객을 만족시키는 방법 및 조건이 필요합니다. 이런 기준이 되는 것이 고객의 **요구사항**입니다. 요구사항이란 **software가 올바르게 동작하기 위해서 수행하는 하나의 일**이라 정의될 수 있습니다. 요구사항을 수집, 분석을 통해 software의 목적과 범위를 결정할 수 있습니다.

Use case 작성

좋은 요구사항은 software가 고객이 기대한대로 동작하도록 합니다. 고객이 제시한 요구사항은 Use case를 통해 정리될 수 있습니다. 즉 고객의 특정한 목표를 달성하기 위해 software가 **무엇을 해야 하는지**를 순차적으로 기술한 내용을 바탕으로 software가 실행되는 일련의 scenario와 도식으로 표현된 Use case diagram을 작성하고 이를 통해 **software의 범위와 구동 내용을 파악**하여 software를 설계하고 개발하게 됩니다.

Use case가 **고객의 요구 사항을 잘 담고 있는지 확인**을 통해서 고객이 잊고 얘기하지 않았던 부분이나 software에 새로이 추가되어야 할 점, 빠져있었거나 불완전했던 요구사항들을 분명하게 명시합니다. 요구사항은 끊임없이 변화하므로 Use case의 확인은 꼭 필요합니다.

완성된 Use 를 가지고 클래스 후보를 찾아내어 각 클래스의 속성, 오퍼레이션 등 개념적인 클래스 다이어그램을 작성합니다. 후에 클래스 사이에 관계를 찾아 UML을 완성합니다.

Design

OOAD를 하는 이유도 결국 **견고한 software를 작성**하는데 있습니다. 견고한 software란 유지보수와 재사용이 가능하고 확장이 용이한 software를 말합니다. 고객의 요구사항은 고정적이지 않습니다. 잘 design된 software는 변경과 확장이 쉬워 항시 변하는 요구사항을 수용하는데 용이합니다. 또한 프로젝트에 참여하는 모든 사람들이 목표로 하는 software를 잘 이해할 수 있게하여 시간과 비용을 절약할 수 있습니다.

이런 유연한 software를 만들기 위하여 OOAD에서는 **OO(Object Orient) 원리**와 **디자인 패턴**을 이용하여 software를 design 합니다. 기본적으로 캡슐화와 상속 같은 OO 원리를 이용하여 software를 유연하게 만듭니다. 또한 각 class는 class당 하나의 기능만 수행하도록 가능한 작은 범위를 유지해야 합니다. 그 외 OO 원리는 아래와 같은 것들이 있습니다.

디자인 패턴이란 남이 수많은 **시행착오를 겪으며 구축해 놓은 좋은 설계**들입니다. 이미 검증된 것 인만큼 이를 인용해서 현재 만들어야할 시스템(혹은 software)을 디자인하면 그만큼 좋은 디자인을 할 가능성이 높아집니다. 어떤 디자인 패턴을 이용할지는 전적으로 개발하는 사람에게 달려있으며 같은 software라도 어떤 디자인 패턴을 이용하느냐에 따라 그 효율성의 차가 나므로 초기 디자인 단계에서 적절한 디자인 패턴을 선택하는 것이 중요합니다.

Other Object Orient principle

변하는 것은 **캡슐화**한다.
구현에 의존하기보다는 인터페이스에 의존하도록 코딩한다.
각 클래스의 변경 요인은 오직 하나여야 한다.
클래스는 **행동과 기능에 관한 것**이다.

Design Pattern

adapter pattern, MVC pattern, decorator pattern
singleton pattern, moment pattern, bridge pattern
.....

Development

디자인까지 끝났으면 이제 본격적으로 software를 만드는 일이 남았습니다. 개발을 하는 방법은 두 가지로 나누어서 볼 수 있습니다. Feature driven development와 Use case driven development입니다.

Feature driven development

- Application의 특정한 특징에 집중
- Application 기능 중 고객이 원하는 기능의 한 조각을 선택하고 그것이 완성될 때까지 개발
- Application의 특정한 특징을 선택하고, 완성될때까지 그 특징을 계획, 분석, 개발하는 것. 한번에 하나의 특징을 작업하고 그리고 나서 반복하여, 어플리케이션의 기능을 완성할 때까지 한번에 하나씩 특징들을 구현한다.

Use case driven development

- 특정한 흐름에 집중

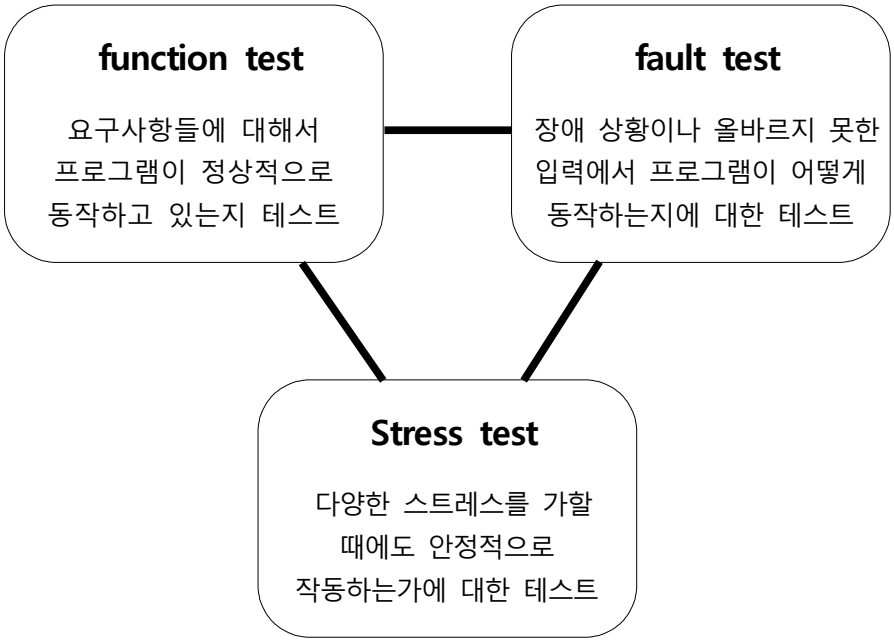
- application에서 하나의 완전한 경로를, 명확한 시작에서 끝까지 선택하여 코드로 구현

- Use case의 scenario를 선택하여 scenario가 완성될 때까지 코드를 작성하는 것.

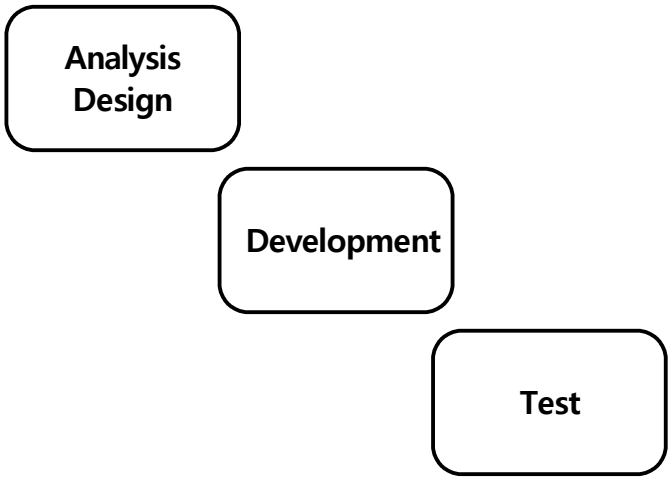
- Use case 내에서 하나의 scenario를 완성하는 작업을 한다. 그 다음 또 다른 scenario를 선택하여 작업하며, use case의 모든 scenario가 완성될 때까지 반복한다. 그리고 나서, 다음 use case를 선택하여 위의 작업을 하며, 모든 use case가 작동할 때까지 반복한다.

Coding 중에 적절한 주석은 해당 code를 분석하고 이해하는데 큰 도움이 됩니다. 다만 지나친 주석은 오히려 code의 가독성을 떨어뜨리므로 적절하게 사용하여야 합니다. 또한 의미에 맞는 변수 이름 사용, 헝가리안 표기법 준수 등으로 개발자간에 code 내용 공유를 도와주므로 사소해 보이는 이런 조그만 것들도 development 단계에서는 큰 영향을 끼칩니다.

Development 단계에서는 코드를 작성하는 일뿐만 아니라 코딩 이 후 만들어진 software를 테스트하여, 동작하는 것을 확인하는 것까지 해야 합니다. 다음과 같은 테스트를 하게 됩니다.



software가 완성될 때까지 -즉, 오작동 없이 software가 기능을 할 때까지- 각 행위, 각 특징, 각 Use case, 각 문제점에 대해 코딩-테스팅-수정 과정을 반복해야 합니다. 따라서 OOAD 개발 과정에서 다음 3단계의 iteration이 생기게 됩니다.



좋은 software는 iteration을 통하여 완성됩니다. Use case driven development의 하나의 scenario든 Feature driven development의 하나의 특징이든 간에 구현을 하면서 분석과 설계를 하고, 다시 이를 되풀이 합니다. 반복을 할 때마다 평가를 하고, 변경이 필요하면 변경을 합니다.

다른 산업과는 달리 software는 실제 구현이 되었다고 수정이 불가능한 것은 아닙니다. 또한 구현 중에도 software의 수정이 비교적 쉽고 빈번히 일어나는 편입니다. 하지만 그렇다고 잦은 수정을 '당연'하게 생각해서는 안됩니다.

analysis/design 단계에서 변경이 development 단계에서의 변경보다 비용이 상대적으로 적습니다. 그리고 test를 거치고 software가 배포가 될 때에 변경은 천문학적인 비용의 손실을 가져올 수 있습니다. 따라서 잘 설계를 한 후에 실제 구현에 들어가는 것이 바람직합니다.

Sequence of OOAD development - part Ⅱ

이제부터 앞서 설명했던 OOAD 개발 흐름을 직접 간단한 software를 개발하는 과정을 통해 '어떻게' 적용되는지 살펴보도록 하겠습니다.

Mission

멩크스

종족 전쟁 전후 분석 결과, 전장에 투입된 자치령 병력의 37%가 중상을 입고 후송 치료를 받은 것으로 나타났다. 최전선에서 전투를 벌이고 있어야 할 병사들 상당수가 야전 병원에 누워 있었다는 뜻이다. 이는 자치령 지도부에 경종을 울리기 충분한 수치였고, 멩스크 황제의 특별 명령에 따라 **수송선 재설계**가 명해졌네. 수송선에 의료 기술을 접목, **수송과 치료라는 두 가지 임무**를 효율적으로 처리함으로써 자치령 해병들이 부상을 당하더라도 더 오래 전장에 남아 임무를 수행할 수 있게 될 것이네. 이 혁신적인 접근법은 금세 우모자 보호령과 켈모리안 연합에서도 채택되었으니 어서 빨리 새로운 방식의 수송선을 개발하도록 하게.

요구사항 수집 | Use case | Design | Development | Out

멩스크 황제로부터 기존 수송선을 개조하여 의료 기능이 추가된 수송선을 개발하라는 명령이 들어왔습니다. 개발하기에 앞서 황제가 원하는 프로그램의 모습을 그려야 합니다. 우선 우리가 개발해야 할 것이 **치료 기능이 추가된 수송선**임을 알고 있으므로 이것이 어떻게 돌아갈 것인가를 짐작해봅니다.

Medical Dropship
What we are thinking

1. 수송선은 비행기 모양이다.
2. 수송선 내부는 2개의 방이 있어 부상자와 정상인을 구분한다.
3. 최대 인원은 한 분대 병력인 12명으로 한다.
4. 수송기 내부에는 의료 기기가 추가되어 해병들의 치료가 가능하다.

하지만 이것은 언제까지나 우리만의 생각일 뿐, 실제 황제가 원하는 수송선은 아닐 수도 있습니다. 그가 원하는 수송선을 만들기 위하여 첫 번째로 그에게서 수송선을 어떻게 만들어야할지 그의 요구사항을 들어야 합니다.

요구사항을 수집 하는 방법으로 **인터뷰, 설문조사, 브레인스토밍, (4C/SWOT)** 등의 문제분석기법 등이 이용됩니다. 이런 여러 기법 중 우리는 황제를 직접 찾아가 인터뷰를 해봅니다. 인터뷰는 사용자로부터 요구사항을 직접적으로 들을 수 있는 방법입니다.

인터뷰를 하기 사전에 인터뷰 대상자를 선정하고 인터뷰 일정 및 계획을 세웁니다. 인터뷰 대상자는 실제 개발해야할 software에 대해 알고 있는 사람으로 선정해야 보다 정확한 요구사항을 얻을 수 있습니다. 황제가 명령을 내린 만큼 우리는 그의 비서실장보다는 그로부터 가장 정확한 요구사항을 들을 수 있을 겁니다.

우리 : 수송선에 어떤 기능을 원하시나요?
황제 : 1. 수송선은 비행기 보다는 헬리콥터 모양이었으면 좋겠어
2. 그리고 내부에 칸막이는 3개정도는 있어야 좋지 않겠어??
3. 최대 탑승 수는 의료진을 제외한 해병 12명이면 충분해.
4. 기왕이면 빠른 수송을 위해 음속으로 만들어 주게나.

황제가 원하는 수송선의 모습을 얻었습니다. 원래 우리가 생각했던 것과는 차이가 있군요. 그런데 요구사항 중에 2번 요구사항 중 칸막이 얘기는 무슨 뜻인지 애매합니다. 이 부분은 황제에게 다시 물어봐야겠습니다. 또 4번 요구사항은 지금 우리에게 주어진 예산으로는 구현하기 버겁습니다. 황제에게 예산에 맞는 기능으로 대체하도록 요구하기로 합니다.

Medical Dropship
Requirement List

1. 수송선은 헬리콥터 형으로 제작한다.
2. 내부에 칸막이를 이용하여 4개의 방을 만든다.
3. 탑승 인원은 (의료진을 제외한) 최대 해병 12명
4 음속 기능 대신 자동 진단 기능을 추가하여 빠르기보다 치료 효율성을 강화한다.

이렇게 인터뷰를 통해 얻은 요구사항을 분석하고 실제 구현이 가능한 요구사항인지 확인을 하고 user의 요구사항을 명확히 함으로써 우리가 개발해야할 software의 범위를 정할 수 있습니다. 또한 software의 범위 결정은 프로젝트의 성공을 좌지우지하는 중요한 요인이자 user가 원하는 software를 개발해 줄 수 있는 효과적인 방법입니다.

요구사항 수집	**Use case**	Design	Development	Out

황제의 요구사항 리스트를 가지고, 좀 더 구체적으로 software가 어떤 일을 해야 하는지 일련의 순서로 작성을 해봅니다. 즉, use case를 작성합니다. 수송선의 기능대로 를 하려고 할 때의 상황을 차근차근 적어보겠습니다.

Medical Dropship
Use case

1. 수송선이 착륙한다.

2. 해병이 수송선에 탑승한다.

3. Medical Dropship 내에서 해병의 부상 여부를 판단한다.
3-1. 부상당하지 않은 해병은 수송선 내 sector 1, 2로 이동한다.
3-2. 부상당한 해병은 sector 3, 4로 이동한다.

4. 해병들의 탑승이 완료되면 수송선은 이륙한다.

5. 수송선이 이동하는 동안 의료 지원 여부를 판단한다.
5-1 1,3 sector 내에 해병들이 있으면 수송선 내 의료진에게 치료를 받는다.
5-2 2,4 sector는 해병 유,무에 상관없이 의료지원을 하지 않는다.

6. 도착지에 도착하면 해병들을 내리고, 다음 지령을 기다린다.

이 use case는 수송선이 수송할 때 어떻게 동작하는지 설명하고 있습니다. use case는 user의 목표가 이뤄지면 끝나게 됩니다. 그리고 위 use case에서 시작부터 끝까지 일련의 순서대로 동작하는 흐름을 묶으면 scenario가 됩니다.

그런데 여기서 잠깐, 우리가 작성한 use case가 황제의 요구사항을 다 들어주고 있는지 살펴 봐야 됩니다. 요구사항과 use case를 비교해보면 황제가 원하는 것은 다 만족시키고 있습니다. 하지만 5-2번에서 1, 3 sector에 3번 과정에서 잘못된 진단으로 부상당한 해병이 들어올 경우에 부상당한 해병을 치료하지 못할 수도 있네요.

우리의 software는 '실제 상황'에서 돌아간다는 것을 생각하며 use case를 수정합니다. 비록 development 단계에서 test를 해서 문제를 잡아낼 것이지만 analysis 단계에서도 문제가 일어날 경우를 생각하여 use case를 작성하고 수정해야합니다.

Medical Dropship
Use case (modified)

1. 수송선이 착륙한다.

2. 해병이 수송선에 탑승한다.

3. Medical Dropship 내에서 해병의 부상 여부를 판단한다.
3-1. 부상당하지 않은 해병은 수송선 내 sector 1, 2로 이동한다.
3-2. 부상당한 해병은 sector 3, 4로 이동한다.

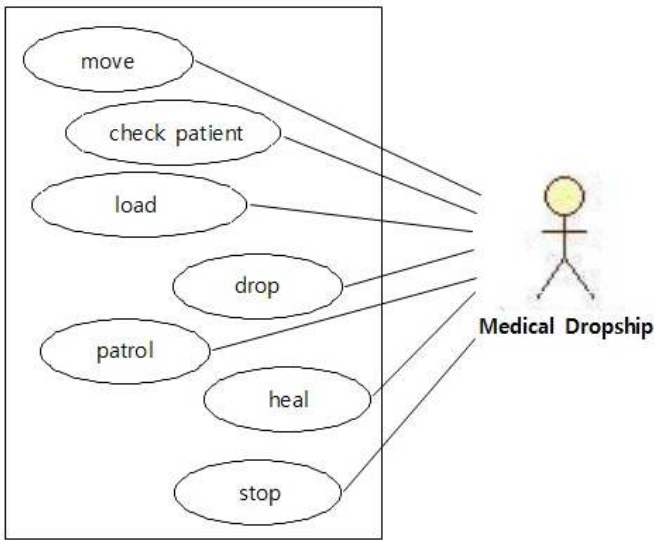
4. 해병들의 탑승이 완료되면 수송선은 이륙한다.

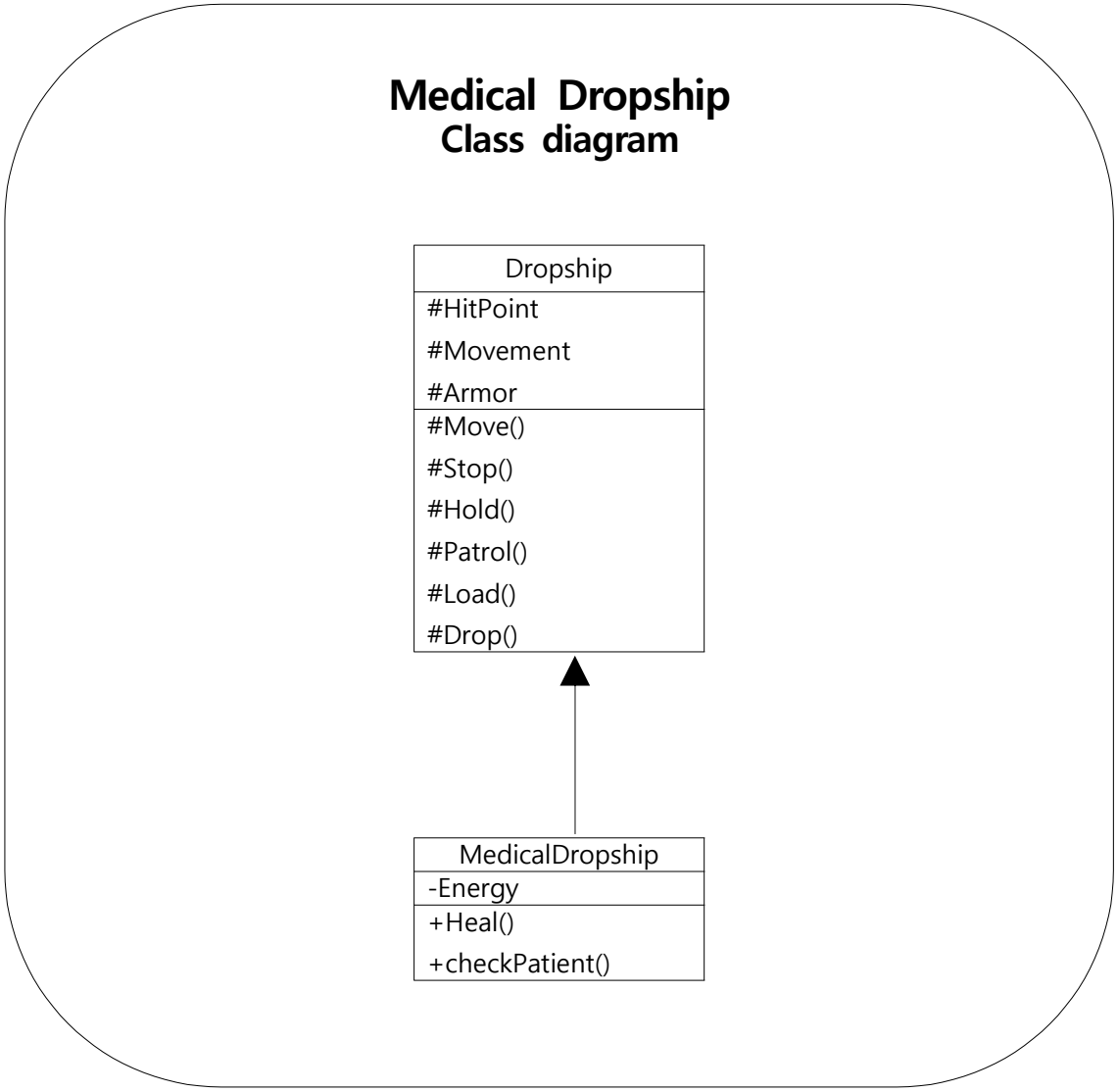
5. 수송선이 이동하는 동안 의료 지원 여부를 판단한다.
5-1 1,3 sector 내에 해병들이 있으면 수송선 내 의료진에게 치료를 받는다.
5-2 2,4 sector의 해병은 의료진이 다시 한 번 진단을 한다.
5-2-1 부상당한 해병이 있으면 sector 3, 4로 보내고 치료를 실시한다.
5-2-2 나머지 해병은 의료지원을 하지 않는다.
수정 부분

6. 도착지에 도착하면 해병들을 내리고, 다음 지령을 기다린다.

여기까지 use case가 작성 완료되면 use case diagram을 그립니다. 그리고 이를 토대로 class diagram을 그리고, class간의 관계를 이어줍니다.

Medical Dropship
Use case diagram





전체적인 software의 그림과 class diagram도 그렸습니다. 이제는 OOAD의 목적에 맞게 이 software가 확장이 용이하고, 재사용이 가능하게끔 견고해지도록 design을 해봅시다. 지금 우리는 Dropship class에서 상속받아 MedicalDropship class를 만들었습니다. MedicalDropship에는 기존 Dropship이 가지고 있는 기능 그대로를 가지고 있고, 추가로 Dropship에는 없는 기능도 추가되었습니다. 기존의 Dropship class를 **확장**해서 새로운 class를 만들었고, 언제든지 Dropship class에서 또 상속을 받아 다른 기능을 추가한 새로운 형태의 dropship을 만들 수 있으므로 **재사용**에 용이합니다.

Dropship class와 의료 기능을 가진 운송수단을 객체화한 MedicalVehicle class를 다중상속 받는 것도 고려해볼 수 있습니다. 이 역시 OO원리를 이용하여 class를 재사용하고 확장하는 것이지만, 여러 가지 난해한 상황을 발생시킬 수 있습니다. 가령 MedicalDropship 객체가 move()를 했을 경우, Dropship class에서 상속받은 move()인지, MedicalVehicle에서 상속받은 move() 인지 판단할 수 없습니다. OO원리를 따르고 있지만 이러한 불안전함 때문에 좋은 design이라 보기 힘듭니다.

이제 본격적으로 coding을 하여 MedicalDropship을 만들겠습니다. 앞서 설명했듯 개발하는 데에는 두 가지 방법이 있습니다. 우리는 기존 Dropship에 '의료 기능' 을 추가한 것으로 이 기능에 초점을 맞춰 feature driven development 방법을 사용합니다.

수송기 내 의료 시설 기능을 구현했다고 끝은 아니죠. testing을 통해서 과연 우리가 구현한 기능이 황제가 원한 기능과 맞는가 확인해 보아야 합니다.

첫 번째로 부상당한 실험자를 태우고 운행 test를 합니다. Medical dropship이 이동하는 동안 실험자의 HP를 확인해보니 우리가 원하는 대로 실험자의 HP가 상승하고 있습니다. Function test는 통과입니다.

다음으로 예외 조건으로 멀쩡한 실험자를 태우고 운행 test를 합니다. 현재 실험자의 HP는 최대이므로 Medical dropship에 탔다고 HP가 더 높아지거나 혹은 낮아지는 현상은 없어야 합니다. 운행 동안 실험자의 HP를 확인해보니 역시나 정상입니다. 이번 Fault test도 통과입니다.

마지막으로, 이번에는 한 명의 실험자가 아닌 최대 인원인 14명의 실험자가 동시에 탑승합니다. 이들 중에는 치료가 필요한 부상자도 있고, 멀쩡한 사람도 있습니다. 지금까지는 한 사람에 따른 test를 했지만, 한 번에 많은 사람을 태웠을 때를 test 합니다. 그런데 부상당한 한 사람의 HP가 채워지지 않습니다. 끝내 마지막 관문인 stress test를 통과하지 못했습니다. 우리는 문제점에 대해 분석하고 다시 구현하고 또 test를 해야 합니다. 물론, Medical dropship이 완벽하게 구현될 때까지 반복을 해야 합니다.

반복된 test를 통해서 문제점들을 모두 잡아내고 마침내 Medical dropship이 완성 되었습니다. 완성된 제품을 황제에게 인계하는 것뿐만 아니라 추가로 계약에 따라 Medical dropship에 관한 manual 제공 및 dropship을 다룰 수 있는 기술 교육, A/S 등과 같은 서비스도 더불어 제공을 해줍니다.

Difference between procedural programming & object oriented programming

지금까지 Medical dropship을 만드는 과정을 통하여 OOAD 개발 과정을 살펴보았습니다. procedural programming과 뚜렷하게 보이는 차이점으로는 OOAD는 data와 수행하는 행동 즉 method가 객체로써 함께 묶여있다는 점입니다. 이에 비해 procedural programming에서는 데이터는 함수 혹은 procedure에 완전히 구분되어져 있습니다. procedure는 data의 입력을 받고 출력을 내보내기만 하는 black box가 되는 것이 이상적이라 할 수 있습니다.

어느 방법이 좋다고 딱 잘라서 좋다고 말하기는 힘들지만 완전히 새로 개발되어지는 software(혹은 system)에서는 OOAD 개발 방법을 사용하는 것이 좋다고 말을 합니다. 다만 기존에 이미 짜여졌던 system들은 - legacy system - OOAD 기법을 이용하여 재개발하기에는 위험 부담이 큽니다. 이미 잘 동작하고 있는 system들인데 굳이 위험을 감수하면서까지 OOAD 기법을 이용하여 새로이 바꿀 필요는 없는 것이죠.

Conclusion - Purpose of OOAD

OOA는 큰 문제를 여러 개의 작은 문제들의 집합으로 봅니다. 작은 프로젝트에서처럼 큰 프로젝트에서도 특징들과 요구사항을 모으는 것부터 시작합니다. 특징은 보통 시스템이 하는 '큰 일' 이지만, '요구 사항' 과 같은 뜻으로 사용될 수 있습니다.

이렇게 작은 단위로 나누어 분석된 요구사항들은 캡슐화같은 OO 원리를 사용하여 유연한 프로그램을 만들 수 있습니다. 기능과 유연한 설계가 완성이 되면 디자인 패턴을 사용하여 프로그램의 디자인을 개선하고, 재사용하기 쉽게 만듭니다.

이렇게 OOAD 방법을 이용하여 개발된 프로그램은 문제가 발생하거나 요구사항의 변경이 생기더라도 그 부분의 객체만을 수정함으로써 빠른 대처가 가능합니다. 또한 제작된 객체는 다른 프로그램에 재사용이 가능하며, 확장을 통하여 비슷한 목적의 다른 프로그램으로도 변형, 확장이 가능합니다.

이런 재사용, 확장, 유지보수의 용이함 덕분에 OOAD 기법이 각광받고 있습니다.

Reference

Head first OOAD - 브랫 맥리프란 외 (O'REILY)
The Object-Oriented Thought Process - 맷 와이스펠드 (Addison-Wesley)
생각하며 배우는 UML 2.0 - 김현남 (영진닷컴)
소프트웨어 공학 - 최은만 (정익사)
Software Engineering (8th) - Ian Sommerville (Addison-Wesley)
wiki Encyclopedia - 'OOAD'