**객체지향 분석설계 Self Study**

2016156007

소프트웨어학과

김영민

내용

[**제 1장. UML의 이해** 1](#_Toc57474316)

[**1.1 UML 용도와 특징** 1](#_Toc57474317)

[**1.2 객체지향 모델링** 3](#_Toc57474318)

[**1.2 Question** 6](#_Toc57474319)

[**제 2장 UML 구성요소와 뷰** 7](#_Toc57474320)

[**2.1 UML의 구성** 7](#_Toc57474321)

[**2.2 UML의 뷰** 12](#_Toc57474322)

[**2.3 UML의 특성** 13](#_Toc57474323)

[**2.4 Question** 14](#_Toc57474324)

[**제 3장 유스케이스 다이어그램** 15](#_Toc57474325)

[**3.1 유스케이스 다이어그램의 구성요소와 표현** 15](#_Toc57474326)

[**3.2 유스케이스 다이어그램의 관계** 17](#_Toc57474327)

[**3.3 유스케이스 다이어그램의 단계별 모델링** 19](#_Toc57474328)

[**3.4 유스케이스 다이어그램의 모델링 연습** 20](#_Toc57474329)

[**3.5 유의사항** 20](#_Toc57474330)

[**3.6 Question** 22](#_Toc57474331)

[**제 4장. 클래스 다이어그램** 23](#_Toc57474332)

[**4.1 클래스 다이어그램의 구성요소와 표현** 23](#_Toc57474333)

[**4.2 클래스 다이어그램 관계** 26](#_Toc57474334)

[**4.3 클래스 다이어그램의 단계별 모델링** 28](#_Toc57474335)

[**4.4 클래스 다이어그램을 효과적으로 작성 방법** 28](#_Toc57474336)

[**4.5 Question** 29](#_Toc57474337)

[**제 5장. 순차 다이어그램** 30](#_Toc57474338)

[**5.1 순차 다이어그램의 구성요소와 표현** 30](#_Toc57474339)

[**5.2 순차 다이어그램의 단계별 모델링** 32](#_Toc57474340)

[**제 6장. 통신 다이어그램** 34](#_Toc57474341)

[**6.1 통신 다이어그램 개요** 34](#_Toc57474342)

[**6.2 통신 다이어그램의 구성 요소와 표현** 35](#_Toc57474343)

[**6.3 통신 다이어그램 모델링 (각 단계의 모델링 순서 암기)** 36](#_Toc57474344)

[**6.4 통신 다이어그램 작성 방법** 37](#_Toc57474345)

[**6.5 순차 다이어그램 단계별 모델링** 37](#_Toc57474346)

[**6.6 통신 다이어그램 작성시 주의 사항** 37](#_Toc57474347)

[**6.7 Question** 38](#_Toc57474348)

[**제 7장. 활동 다이어그램** 39](#_Toc57474349)

[**7.1 통신 다이어그램 개요** 39](#_Toc57474350)

[**7.2 활동 다이어그램 용도** 41](#_Toc57474351)

[**7.3 활동 다이어그램 작성 순서** 42](#_Toc57474352)

[**7.4 활동 다이어그램 작성시 주의사항** 43](#_Toc57474353)

[**7.5 Question** 43](#_Toc57474354)

[**제 8장. 상태 다이어그램** 44](#_Toc57474355)

[**8.1 상태 다이어그램 표현과 용도** 44](#_Toc57474356)

[**8.2 상태 다이어그램 구성 요소** 45](#_Toc57474357)

[**8.3 상태 다이어그램 표현과 용도** 45](#_Toc57474358)

[**8.4 상태 다이어그램 표현과 용도** 46](#_Toc57474359)

[**8.4 Question** 47](#_Toc57474360)

[**제 9장. 컴포넌트 다이어그램** 48](#_Toc57474361)

[**9.1 컴포넌트 다이어그램 표현과 용도** 48](#_Toc57474362)

[**9.1 컴포넌트 다이어그램 작성 순서** 50](#_Toc57474363)

[**9.3 Question** 51](#_Toc57474364)

[**제 10장. 배치 다이어그램** 52](#_Toc57474365)

[**10.1 배치 다이어그램 표현과 용도** 52](#_Toc57474366)

[**10.2 배치 다이어그램 작성 순서** 53](#_Toc57474367)

[**10.2 배치 다이어그램 작성 시 주의사항** 53](#_Toc57474368)

[**10.4 Question** 53](#_Toc57474369)

[**제 11장. 패키지 다이어그램** 54](#_Toc57474370)

[**11.1 패키지 다이어그램 표현과 용도** 54](#_Toc57474371)

[**11.2 패키지 다이어그램 작성 순서** 56](#_Toc57474372)

[**11.3 패키지 다이어그램 작성시 주의사항** 57](#_Toc57474373)

[**11.4 Question** 57](#_Toc57474374)

# **제 1장. UML의 이해**

## **1.1 UML 용도와 특징**

|  |  |
| --- | --- |
| 시작이 반이다 | 이 과목에서 무엇을 얻을 것인지 어떤 공부를 할 것인지 결정할 것. 객체지향은 나무를 보지 않고 숲을 보는 것. view point 와 concept가 중요하다.  마부작침 -> 도끼를 갈아서 바늘을 만든다. |
| UML의 탄생과 특징 | 정보 공학의 대가들이 객체 관리 그룹에서 여러 표기법을 통합하여 UML을 발표하였음  객체 지향 모델링 언어를 여러 사람들의 방법론을 합쳐 UML을 만듬 |
| OMT | GE 프로젝트를 다수 수행해본 경험을 통해 object Oriented Modeling and Design을 출간 -> 다양한 모델로 전체 모델링을 쉽게 한다.  럼버 -> 다양한 모델이 중요하다. (객체 모델, 동적 모델, 기능 모델) |
| Booch Method | 어플리케이션 기반의 분석 설계를 만든다.  시스템 아키텍처에 초점을 두어 구성 요소간 관계와 상호작용을 정의하였음 |
| OOSE | **유스케이스**를 중심으로 접근법을 두었다.  위에 것들은 알 필요는 없고 유스케이스가 중요 |
| UML | Unified Modeling Language의 약자로 **특정 기호를 가지고 표현하는 것** 소프트웨어를 시각화, 명세화, 생성하고 문서화 하기위한 표준화된 언어를 말함  시스템 개발자가 자신의 비전을 구축하고 반영할 수 있음  **다른 사람들과 효율적으로 공유**할 수 있는 구조 |
| UML이 제공하는 표준화된 다이어그램 | **-> 유스케이스 다이어그램**  **-> 클래스 다이어그램**  **-> 순차 다이어그램**  **-> 통신 다이어그램**  **-> 활동 다이어그램**  **-> 상태 다이어그램**  **-> 컴포넌트 다이어그램**  **-> 배치 다이어그램**  **-> 패키지 다이어그램** |
| UML 특징 | UML은 **시각화 언어** -> 소프트웨어 개념 모델을 시각화 하여 **명확히 정의된 표준화된 다이어그램 제공**  UML **명세화 언어** -> 설계 단계 각 과정에서 필요한 모델을 **정확하고 완전하게 명세화**  UML은 **구축(코딩) 언어** -> **다양한 프로그래밍 언어로 표현 가능, 프로그램 코드로 자동 변환 가능, 역공학 가능**  **StarUML 권장**  UML은 **문서화 언어** -> **설계한 내용을 자동으로 문서화** 가능 |
| UML과 모델링 | 개발하고자 하는 프로그램을 **시각적**으로 표현하는 것으로  -> 의뢰자의 요구에 맞게 **쉽게 수정** 가능  -> 따라서 **유지보수** 기간을 줄여 **생산성**을 높이는 결과  -> 코드보다는 모델을 보여줌으로 써 시각화에 용이 |
| Model | 현실을 단순화시킨 것, 시스템의 청사진을 제공 |
| 모델을 만드는 이유 | **개발자의 이해 증진을 위함** -> 구체적으로 보여줌으로써 모델 이해를 더욱 쉽게 할 수 있다. |
| 모델링을 하는 4가지 목적 | **가시화** -> 모델을 원하는 모습으로 가시화  **명세화** -> 모델은 시스템의 구조와 행동을 명세화 할 수 있음  **틀 제공** -> 모델은 시스템을 구축하는 틀을 제공  **문서화** -> 모델은 우리가 결정한 것을 문서화 가능 |
| 모델의 규칙 | **표준화**된 표현 방법을 사용해야함  고객과 사용자가 **쉽게 이해**할 수 있어야 함  엔지니어가 시스템에 대하여 **통찰력**을 가질 수 있게 해야함  **추상**을 제공 해야함(상세) |

## **1.2 객체지향 모델링**

|  |  |
| --- | --- |
| 객체지향 개념 | 구조적 방법의 문제점을 극복하고 인간이 사고하는 방식대로 프로그램을 개발하려는 노력으로부터 탄생함  실체 및 개념들을 객체라는 독립된 단위로 구성하고 메시지메 통하여 상호작용함으로써 전체 시스템이 운영되는 개념들을 객체라는 독립된 단위로 구성함  **유지 보수와 결합도를 낮추는 핵심적인 idea -> 객체지향 개념** |
| 객체 | **사물이나 개념을 의미(현실 세계에 존재에 존재하는 모든 것)**  **클래스에 찍어 내는 것** |
| 클래스의 인스턴스 | 객체는 클래스에 정의된 대로 생성 |
| 객체의 구성 | **상태**: 해당 객체의 정보  **행동**: 해당 객체의 행동이나 반응을 정의하는 부분  **정체성**(이름): 객체를 다른 객체와 구분 짓는 이름  특성: 속성 + 오퍼레이션 (이 두가지를 가지고 있다는 의미) |
| 클래스 | **오브젝트를 만들어 내는 양식.**  **객체의 설계도 또는 틀** |
| 클래스의 용도 | 객체를 생성하는데 사용. 객체는 클래스에 정의된 대로 생성 |
| 메시지 | **객체 간의 상호작용 수단**  한 객체가 다른 객체에 특정 작업을 요청하는 신호  메시지를 보내는 객체는 송신 객체  메시지를 받아서 동작을 수행하는 객체는 수신 객체 |
| 추상화 | **특정 측면을 강조**하여 나타내는 것  **추상화**가 되어야 만 **모델**을 만들 수 있다.  **실체화: 추상화한 모델링을 프로그램으로 구현(코딩까지 구현)**  **추상화 -> 실체와** |
| **캡슐화** | 내부의 내용을 다른 사람들에게 알려주지 않고 인터페이스로 사용 방법만 알려줘 프로그램의 **유지보수**와 **보안**에 장점이 있음 |
| **상속(중요)** | **프로그램을 쉽게 확장할 수 있도록 도와주는 수단**  **정보를 공개하고 재사용 하는 개념**  **상세화:** 상위 클래스의 속성을 상속받아 하위 클래스에서 실체화하는 관계  **일반화**: 하위 클래스의 공통 특성을 추상화하여 상위 클래스로 정의하는 것 |
| **다형성** | 여러 클래스에 **같은 이름의 함수가 존재하지만** **동작은 다르게 수행**하는 것  **이름은 같지만 동작은 다르다**  하위 클래스에서는 그들 만의 **고유한 속성과 오퍼레이션 재정의 필요**  객체 지향 언어에서 **메서드 오버라이딩** 방식으로 구현 |
| 추상 클래스 | 추상 클래스에는 클래스의 명칭과 메서드는 있으나 메서드의 처리 내용은 없음  추상 클래스는 추상 메서드 외에 일반적인 속성과 메서드를 가질 수 있음  메서드의 다형성 지원 |
| **인터페이스** | **상수와 추상 메서드만 가지는 것**  여러 개의 인터페이스로부터 상속받을 수 있기 때문에 다중 상속의 기능을 제공  속성을 가질 수 없으며, 메서드의 구현도 정의할 수 없음 |
| **모델링** | 시스템을 구축할 때 **개발자가 고민하고 결정하는 모든 활동**  구현 단계 이전의 요구 사항 정의, 분석, 설계에서 수행하는 활동 |
| 모델링 방법 | **부치 방법론** -> **뷰 중심**, 시스템을 몇 개의 뷰로 분석  **야콥슨의 OOSE** -> 유스케이스를 강조한 방법론 -> 큰 규모의 시스템에 유리  **럼바의 OMT** -> 객체 모델, 동적 모델, 기능 모델 세가지 사용 |

## **1.2 Question**

1. UML의 정의가 아직 완벽하게 숙지 되지 않았습니다. 특정 프로그램을 지칭하는 것이 아닌 다이어그램의 묶음인가요?

2. StarUML의 장점이 궁금합니다.

3. 상속의 일반화에서 하위 클래스의 공통 특성을 추상화하여 상위 클래스로 정의한다는 말이 잘 이해가지 않습니다.

4. 예전부터 의문이었던 부분이었는데 실무 과정에서 추상 클래스와 인터페이스가 유사한 부분이 많은데 주로 사용하는 것은 인터페이스인가요?

5. 부치 방법론에서 뷰를 중심으로 분석한다는 말이 잘 이해가지 않습니다.

# **제 2장 UML 구성요소와 뷰**

|  |  |
| --- | --- |
| 멀리 그리고 크게 보자 | 실력을 산으로 비유했을 때 위로 올라갈수록 상상할 수 없는 부분을 볼 수 있다. 고로 멀리 보고 크게 보자. 나무를 보지 말고 숲을 보자.  아는 사람은 좋아하는 사람을 이길 수 없고 좋아하는 사람은 즐기는 사람을 이길 수 없다.  내가 선택한 일이니 이왕 즐기면서 하자. |

## **2.1 UML의 구성**

|  |  |
| --- | --- |
| UML의 구성요소 | 3가지  사물: UML은 기본 요소를 구성  -> **정적 사물, 동적 사물, 주해 사물**  관계: 사물 간의 관계를 나타냄  -> **의존 관계, 연관 관계**, 일반화 관계, 실체화 관계  다이어그램: 사물과 관계를 도형으로 표현  사물과 관계를 합치면 다이어그램이 된다.  **-> 유스케이스 다이어그램**  **-> 클래스 다이어그램**  **-> 순차 다이어그램**  **-> 통신 다이어그램**  **-> 활동 다이어그램**  **-> 상태 다이어그램**  **-> 컴포넌트 다이어그램**  **-> 배치 다이어그램**  **-> 패키지 다이어그램** |
| 정적 사물 | 모델의 구조, 즉 개념적/물리적 요소를 표현하는 명사  -> 클래스, 인터페이스, 통신, 컴포넌트, 패키지, 노드 등 |
| 클래스 | 동일한 속성 오퍼레이션, 관계 의미를 공유하는 객체를 기술한 것 **양식**  -> 클래스는 UML에서는 사각형으로 정의해서 3개로 분류한다.  -> 사각형 안에 이름, 속성, 오퍼레이션 |
| 인터페이스 | **클래스에서 오퍼레이션만 가진 것**  **다수의 클래스에서 통신 가능**  **하나의 클래스가 여러 개에 통신하는 것이 통신**  서비스를 명세화 하는 **오퍼레이션(메소드)**을 모아 놓은 것  특정 클래스나 컴포넌트의 전체 또는 일부분만의 행동을 나타냄  원으로 표현하고, 인터페이스 명을 아래에 표시하거나 클래스 형식으로 표현, 스테레오 타입으로 <<interface>>를 사용 |
| 통신 | 요소들을 **교류**하는 것  -> 동작과 구조에 중요하고 **클래스 하나가 다수의 통신에 참여 가능**  -> 실선으로 된 사각형으로 표현 |
| 컴포넌트 | 시스템의 물리적이고 대체 가능한 부분  -> 서로 다른 논리 요소를 **물리적으로 패키지화** 한 것  -> 탭이 달린 직사각형으로 표시하여 이름을 안에 넣음 |
| 패키지 | 많은 요소들을 그룹으로 묶어주는 것  -> **정적 사물, 동적 사물**도 하나의 패키지 내에 들어간다.  탭이 달린 폴더로 표현, 보통 이름만 쓰지만 가끔 내용도 기입 |
| 노드(node) | 실행할 때에 존재하는 물리적 요소  **전산 자원을** 나타낸다.  **컴포넌트가 노드에 존재**할 수 있고 노드에서 노드로 이동 가능 |
| 동적사물 | 모델의 동적인 부분을 동사로 표현, 시간과 공간에 따른 행동을 나타낸다.  -> 교류  -> 유스케이스  -> 상태 머신 |
| 교류 | **객체들 간에 주고받는 메시지로 구성된 동작**  **->** 메시지는 직선으로 나타내고, 항상 오퍼레이션 이름을 포함. |
| **유스케이스(중요)** | 유스케이스는 **사용자들이 시스템에 요구하는 함수이다**. 사용자가 시스템에 요구하는 기능을 뜻함 **액터가 시스템에서 요구하는 하나하나의 기능을 유스케이스라고 함** |
| 상태 머신 | **형광등 티비 이런 것도 상태 머신 켜졌다 꺼졌다**  **순서적으로 어떠한 삳태에서 바뀌는 것을 보여준다.**  **객체의 상태와 상태의 변화 순서를 기술,** 특정 행동을 하나의 상태로 지정한다.  상태 머신의 요소  -> 상태 전이: 상태에서 다른 상태로의 흐름  -> 사건: 전이를 유발하는 것  -> 활동: 전이에 따른 응답  **기계를 제어하는 것은 전부 상태 머신으로 제어한다.** |
| 주해사물 | 모델링에는 참여하지 않고 정보를 좀더 표시해 주어서 UML **모델을 설명해주는 부분**.  요소에 대해서 **부연설명**을 해주는 것  -> 노트 |
| 노트 | 첨부되는 제약과 주석을 나타내기 위해 사용한다.  모서리가 접힌 직사각형으로 표현 |
| 의존 관계 | 두 사물 간의 의미적 관계  -> 한 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우에 주로 나타냄  (박스 모양은 외울 필요는 없으나 이해할 것) |
| **연관 관계** | (박스 모양은 시험에 안 낸다.)  의존하는 방향으로 화살표를 한다.  연관은 구조적 관계  -> 어느 한 사물 객체가 다른 사물 객체와 연결을 의미  -> 실선으로 표현  **이름:** 연관은 이름을 가질 수 있음. 관계의 의미를 설명하기 위한 이름이 있어야 한다. 정확한 방향성 또한 있어야 한다.  **역할**: 이 두개에 대한 클래스에 이름이 있다면 각자가 지닌 특별한 역할을 가진다.  클래스가 수행하는 역할을 명시적을 이름 지을 수 있음  **다중성**: 두개의 관계에 몇 개와 몇 개를 해준다 (n: m)  상대 쪽에는 몇 개의 객체가 연결되어 있는지 밝히는 것이 중요할 때가 있음  **집합 연관:** 두 동료 클래스 사이의 구조적 관계를 표현하는데 두 클래스는 개념적으로 같은 수준에 위치.  -> 집합 관계는 전체가 모여 의미가 있지 분리가 되면 의미가 없을 때 -> 집합 관계로 묶어주면 좋다. |
| **일반화 관계** | 좀 더 특수화된 사물  일반화는 자식 객체가 부모 객체가 사용되는 어느 곳에서나 사용될 수 있음  -> 일반화는 주로 클래스와 인터페이스 사이에서 **상속 관계를** 보여주기 위해 사용함  -> 일반화는 실선으로서 속이 빈 큰 화살표로 부모를 향해 그림  **밑에서 위로 올라가는 것이 일반화 관계** |
| **실체화 관계** | 행위만 있는 클래스를 일반화 클래스에 묶어준다.  의존과 일반화의 혼합  인터페이스 티비와 리모컨은 실체화 관계에 있다.    텔레비전과 리모컨은 실체화  사용자는 리모컨에 의존 관계 |
| 다이어그램 | 다이어그램은 **정적 요소와 동적 요소를 그림으로 표현한 것** |
| 시스템의 정적인 부분을 가시화 다이어그램 | -> 클래스 다이어그램  -> 컴포넌트 다이어그램  -> 배치 다이어그램  -> 패키지 다이어그램 |
| 시스템의 동적인 부분을 가시화 다이어그램 | -> 유스케이스 다이어그램  -.> 순차 다이어그램  -> 통신 다이어그램  -> 활동 다이어그램  -> 상태 다이어그램 |
| 클래스 다이어그램 | 클래스, 인터페이스, 통신, 이들의 관계를 통해서 **객체지향을 나타내는 개념** |
| 객체 다이어그램 | **객체와 객체 사이의 관계를 그리고 구조적 상태를 표현**,  시스템의 정적 설계 뷰와 정적 프로세스 뷰를 실제 사례나 프로토타입 사례의 시각으로 보여주는 것 |
| 컴포넌트 다이어그램 | 컴포넌트 사이의 구성과 의존을 나타냄 |
| 배치 다이어그램 | 시스템의 **비구조적 성능**, 부하 등등 배치 프로그램 |
| 패키지 다이어그램 | **여러 모델의 요소를 그룹화** |
| 유스케이스 다이어그램 | 유스케이스와 행위자의 관계를 구조적으로 나타냄 |
| 순차 다이어그램, 통신 다이어그램 | 플로우 차트  교류도의 한 종류로 시스템의 동적 뷰를 다룸  순차 다이어그램: 메시지를 시간적 순서를 강조하는 교류도  통신 다이어그램: 객체 상호관계의 관점, 객체의 구조적 구성을 강조 |
| 활동 다이어그램 | 시스템 내부에 있는 활동의 흐름, 동적 뷰를 보여줌 |
| 상태 다이어그램 | **상태 머신을 나타내며 상태, 전이, 이벤트, 활동으로 구성하여 순차적으로 시간에 따라 바뀌는 행위에 대한 빠른 반응이 필요한 시스템에 유용**  활동 상태: 다른 제어 흐름을 가지는 활동 또는 동작 상태 |

## **2.2 UML의 뷰**

|  |  |
| --- | --- |
| 소프트웨어 아키텍처의 개 view | 이 그림은 반드시 이해할 것 (영상 2\_3 다시 보기) |
| 유스케이스 뷰 | **시스템의 기능 요구사항을 보여주는 관점**  -> 원하는 기능이 “무엇”인지를 정의  -> 다른 뷰를 유도하는 중심 역할  -> 시스템을 하나의 블랙박스로 바라봄  -> 이 뷰를 많이 쓰고 핵심적으로 쓴다. |
| 설계 뷰 | **실제 시스템의 기능 요구사항을 상세적을 보여주는 것**  -> 어떤 클래스, 컴포넌트가 필요  -> 구체적으로 어떻게 하는지 “how”의 개념  -> 클래스와 컴포넌트를 파악하고 기술 |
| 프로세스 뷰 | **시스템 내부의 구조, 즉 클래스와 클래스 사이의 관계, 동작, 상호작용을 중점으로 한다.**  가정해 가면서 어떻게 갈 것인지를 기술  시스템 내부가 어떠한 구조로 되어 있는가  -> 전체의 시스템에 어떻게 상호작용하는지 클래스와 클래스 사이의 관계, 클래스의 행동 및 클래스 사이의 상호 작용에 초점을 맞춤 |
| 구현 뷰 | 실제 행위를 할 수 있는 **구현 형태를 나타내기 위해 모아 노은 것.** 모듈과 그들 사이의 관계, 각종 파일들 간의 의존 관계를 보여주는 뷰 |
| 배치 뷰 | 설치를 어떻게 할 것인지  -> **컴퓨터와 컴퓨터 간의 통신방법에 초점** |
| 여러 가지의 다이어그램을 사용하는 이유 | -> 제대로 설계된 시스템 설계라면 가능한 시점의 모든 다이어그램이 포함되어 있어야 함  -> 각각의 UML 다이어그램은 자신이 나타내고 있는 시점을 하나로 합칠 수 있는 수단을 제공한다.  -> 모든 참여자를 만족시키기 위함 |

## **2.3 UML의 특성**

|  |  |
| --- | --- |
| 명세서 | **표준화된 명세서로 명확한 구문과 의미를 표현**  -> 클래스 이름, 속성, 오퍼레이션 등을 표현하는 방법을 제공 |
| 장식 | 중요 특징을 표현하기 위해 고유한 그래픽 표기 |
| 공통 분할 개념 | 1) 클래스와 객체를 명확하게 분할  클래스: 추상 개념  객체: 추상 개념을 구체적으로 명시  2) 인터페이스와 구현의 분할  인터페이스: 계약  구현: 계약의 구체적인 실현 |
| 확장 방법: 3가지 | 스테레오 타입: <<>>  꼬리표 값: {tag=value}  제약: {} |

## **2.4 Question**

**1. 컴포넌트에 대한 정의를 아직 확실하게 이해하지 못했습니다.**

**2. 컴포넌트가 노드에 존재할 수 있다는 뜻이 잘 이해가 가지 않습니다.**

**3. 상태 머신의 대한 정의를 잘 이해하지 못했습니다.**

**4. 시스템을 개발할 때 반드시 5가지의 뷰를 모두 사용하는지 궁금합니다.**

**5. 공통 분할 개념에서 인터페이스는 계약이라고 하였는데 잘 이해가 가지 않습니다.**

# **제 3장 유스케이스 다이어그램**

|  |  |
| --- | --- |
| 사용자의 한계 능력과 상황에 맞추어 사고 | 고객이 만족하면 전체가 만족한다. 고객을 만족시키기 위해서는 고객에 대한 상황에 맞추어서 생각해야 한다. 나의 중심대로 생각을 하게 되면 고객과의 차이가 생긴다. 따라서 사용자의 한계 능력과 상황에 맞추어 사고하는 방식이 중요하다. 고객 만족의 시작은 자기 스스로의 창의적인 생각이 가능하면 고객이 만족하게 된다. |

## **3.1 유스케이스 다이어그램의 구성요소와 표현**

|  |  |
| --- | --- |
| 유스케이스 | 사용자의 경우의 수. 하나하나가 독립적으로 실행될 수 있는 **시스템의 기능에 대한 단위**를 말한다.  -> 외부 액터와 요구사항이 관련된 것을 유스케이스라고 함  -> (사용자가 시스템에 요구하는)**사용자 관점**으로 정의하여야 한다.  -> **액터**에게 제공하는 사용자 과점의 기능 단위  -> **시스템**이 제공하는 기능 혹은 서비스  -> **구현 방법이나 물리적 환경과는 독립적**.  시스템 내부에서 주고 받는 것은 유스케이스라고 안 한다  액터와 상호작용 하는 것을 유스케이스라고 한다 |
| 액터 | 액터는 행위자. 액터가 행위를 하면 시스템의 처리를 요구하고 정보를 받는다. 액터는 시스템에 한번 이상의 상호작용을 통해서만 의미가 있다.  **사용자와 상호 작용없이 시스템 내부에서 수행되는 기능은 유스케이스가 아니다.** |
| 유스케이스  다이어그램 | 유스케이스로 이루어진 다이어그램. 구성요소(액터, 유스케이스)간의 **전체적인 관계**를 **그림의 형태로 표현**  -> 사용자 관점에서 요구사항을 모아 시스템의 **범위와 기능** 정의  -> 시스템 관점에서 사용자에게 **어떠한 기능을 제공해야** 하는지를 기술  관계자들에게 쉽게 표현하는 것이 중요  어떤 기능을 액터에게 제공하는지 쉽게 알 수 있어 많이 씀 |
| 유스케이스 다이어그램 구성요소 | 액터: 유스케이스에서는 시스템을 사용하는 사용자  유스케이스: 시스템의 행동  subject: 구현하려고 하는 소프트웨어 시스템의 범위 |
| 유스케이스 다이어그램의 필요성 | 모든 것의 표준이 됨 시스템에 사용자가 누구인지, 목적이 무엇인지, 다른 시스템과 연동이 되는지 표현해준다.    여기서 사각형이 개발해야 할 부분인 subject[scope]이다. |
| 유스케이스 다이어그램 사용 목적 | 사용자 정의 -> **Actor**  사용자 요구사항 정의-> **Usecase**  업무 범위 정의-> **Subject**  🡪 **여기 까지가 요구사항 정의**  -> 분석, 설계 작업 기준  -> 사용자와 개발자간 의사소통 원활  -> 테스트의 기준이 됨 |
| 유스케이스 | |  |  | | --- | --- | | **정의** | 시스템의 기능(사용자 관점의 시스템이 제공하는 서비스) | | **표현** | 타원 | | **표기** | 목적어 + 행위 -> 타원안에 표시 | | **의미** | -> 쓰임새를 나타냄 | | **예제** | 우리가 사는 집  -> 의, 식, 주 | |
| 액터(Actor) | 시스템과 **상호 작용하는 사람이나** **시스템이** 수행하는 역할  액터는 우리가 설계하지 않고 구현하지 않는다.  사람 이름 아닌 직책 이름 많이 사용  시스템과 상호 작용하는 외부시스템도 전형적인 액터이다. |
| 액터의 분류 | -> 유스케이스를 가동시키는 액터  -> 시스템의 요청에 따라 정보를 제공하는 외부 액터  -> 시스템이 제공하는 정보를 단순히 수신하는 액터  -> 다른 액터와 시스템 간의 상호작용을 돕는 프록시 역할의 액터(콜센터 직원과 고객) |
| 액터 | |  |  | | --- | --- | | **정의** | 시스템과 상호 작용하는 **사용자 혹은 타 시스템** | | **표현** | stick man 으로 표현 | | **표기** |  | | **의미** | -> 사람/어떤 것  -> 외부적인 것들을 추상화한 것 | | **예제** | -> 손님  -> 은행의 주 전상망 | |

## **3.2 유스케이스 다이어그램의 관계**

|  |  |
| --- | --- |
| 액터와 유스케이스 | 1. 유스케이스 사이의 포함 관계 2. 유스케이스 사이의 확장 관계 3. 액터 사이의 일반화 관계 4. 연관 Association 관계 |
| 유스케이스간 포함 관계 | |  |  | | --- | --- | | **정의** | 한 유스케이스에 반드시 포함되어 수행되는 기능 | | **표현** | 파선이 얇은 머리 화살표 | | **표기** | **공통되는 부분을 밖으로 뽑아내** 유지보수 쉬움 | | **의미** | 특정한 유스케이스가 다른 유스케이스를 포함하고 있는 경우 | |
| 유스케이스간 확장 관계 | |  |  | | --- | --- | | **정의** | 한 유스케이스에서 선택적으로 확장할 수 있는 기능을 의미 | | **표현** | 파선이 얇은 머리 화살표 | | **표기** | 동일한 기능(초록 원)에 **다른 기능이 붙어 있음**  **(갈, 파)**  -> **이 다른 기능들을 한꺼번에 붙여 확장을 시킨다.** | | **의미** | 특정한 유스케이스가 다른 유스케이스를 포함하고 있는 경우 | |
| 포함과 확장의 차이 | 조건에 맞으면 확장관계  항상 포함이 된다면 포함 관계  포함관계는 두개의 관계가 반드시 필수적  확장 관계는 선택적 |
| 액터 사이의 관계 중 일반화 관계 | 액터들이 유스케이스와 **중복하여 관계가 나타나면** **액터들을 통합**하여 일반화 관계로 표현  -> 시스템이 간단해지고 추가 변경이 쉬워진다. |

## **3.3 유스케이스 다이어그램의 단계별 모델링**

|  |  |
| --- | --- |
| 모델링 | 각 단계의 프로세스를 하나로 합친 것을 모델링이라고 한다. |
| 유스케이스 모델링 단계 | **모델링 순서는 반드시 암기**  5단계의 세부적인 5단계로 순서적으로 이루어져 있는 작업이다.  **1단계: 시스템 상황 분석**  -> 문제 정의서, 문제 기술서 작성 (기능 정의)  **2단계: 액터 식별**  -> 행위자와 그들의 책임을 확인  -> 시스템의 주요기능, 지원에 필요한 사람, 유지 관리하는 사람, 필요한 하드웨어 장치, 상호작용하는 다른 시스템, 연결되는 사람 또는 사물은 무엇인지 찾는다.  -> 즉 액터는 사용자, operator, 외부 장치로 구분해서 물어봐서 찾아낸다.  **3단계: 유스케이스 식별**  -> 액터 관점에서 시스템의 기능을 확인  **4단계: 유스케이스 다이어그램 작성**  -> 액터와 유스케이스 관계를 설정  -> 유스케이스에서 <<include>> 의존성이 있는지 평가  -> 유스케이스에서 <<extend>> 의존성이 있는지 평가  -> 액터의 일반화 관계를 찾음  **5단계: 유스케이스 명세서 작성**  -> 유스케이스 명, 액터 명 및 개요를 기술  -> 사전 및 사후 조건과 제약사항들을 식별  -> 작업 흐름과 시나리오 도출  -> 유스케이스 흐름에서 포함이나 확장 유스케이스로 구조화  6단계: 유스케이스 실체화  -> 구현 시스템의 논리적 구성 요소인 클래스 식별 후 통신 관계 파악 |

## **3.4 유스케이스 다이어그램의 모델링 연습**

|  |  |
| --- | --- |
| 재고관리 | 1단계: 시스템 상황 분석  2단계: 액터 식별  3단계: 유스케이스 식별  4단계: 유스케이스 다이어그램 작성  5단계: 유스케이스 명세서 작성  에 따라 순서에 맞춰서 모델링하기 |

## **3.5 유의사항**

|  |  |
| --- | --- |
| 전통적인 실수 | **1. 액터가 없는 유스케이스, 시스템이 유스케이스**  -> 액터와 유스케이스는 반드시 서로 연결되어 있어야함  **2. 사용자 인터페이스의 내용을 너무 상세히 담고 있는 유스케이스**  -> 기능적 요구사항을 담는 것이다.  -> 사용자 인터페이스의 상세한 내용은 담지 않는다.  **3. 너무 낮은 레벨의 유스케이스**  -> 구현에 대한 상세한 내용이 담겨 있는 유스케이스는 적절하지 않음  **4. 전산인의 언어를 사용한 유스케이스**  -> 많은 사람이 이해할 수 있도록 보편적인 용어 사용  **5. 애플리케이션 관점의 유스케이스**  -> 유스케이스는 액터의 관점에서 기술해야 한다.  **6. “~해야 한다.” 식의 표현**  -> ~한다. 대신 사용 |
| 작성에 대한 오해 | **1. 다른 형태의 요구사항 문서를 만들 필요가 없다.**  **유스케이스만 있으면 요구사항을 충분히 반영할 수 있다**  **-> 더 적합한 다이어그램도 존재한다.**  **2. 읽는 사람이 유스케이스의 목적에 대해 이해하기 힘들게 한다.**  **-> 명확하게 유스케이스를 작성해야 한다.**  **3. 유스케이스의 범위에 대해서 혼란스럽게 한다. 범위는 점점 퍼져 나갈 것이고 나중에 리팩토링을 하면 된다. 사용자는 결국 생각을 계속 바꿀 것인데 성가시게 범위를 미리 확정할 필요는 없다.**  **-> 시스템이 어느 정도의 범위를 다루는가를 표현**  **4. 유스케이스 기술서에 비기능적 요구사항과 사용자 인터페이스 디테일을 포함시킨다**  **-> 사용자 인터페이스에 대한 세무 내용을 포함시키면 작업 흐름이 늘어나게 된다.**  **5. 초기 유스케이스 다이어그램에 포함관계와 확장관계를 많이 사용한다. 그러면 유스케이스를 작성 단위의 것으로 나눌 수 있다.**  **-> 반복을 거치면서 보다 상세한 유스케이스를 작성하는 것이 좋다.**  **6. 비즈니스 룰을 정의하는 것에는 관여하지 말라**  **-> 관여해줘야 한다.**  **7. 유스케이스의 작성에 도메인 전문가를 관여하지 말라. 그들은 질문이나 해댈 뿐이다.**  **-> 그 분야의 전문가가 당연히 들어와야 한다.**  **8. 만약 사용자를 유스케이스 정의에 관여한다면 그냥 그렇게 할 것.**  **-> 충분히 이해시키고 유스케이스를 통해 효과를 보라**  **9.** **유스케이스를 한번에 아주 상세하게 만들어라**  **-> 반복적으로 점증적으로 유스케이스 작성이 좋다.**  **10. 유스케이스를 검증하거나 평가하지 말라. 재 작업이나 만들어 낼 뿐이다.**  **-> 중간 중간에 계속 사용자나 도메인 전문가의 피드백이 필요** |

## **3.6 Question**

1. 액터에서 왜 사용자와 상호 작용없이 시스템 내부에서 수행되는 기능은 유스케이스가 아닌 것인지 잘 이해가 가지 않습니다.

2. 다이어그램에서 스테레오 타입이 무엇을 말하는 것인지 용어를 잘 모르겠습니다.

3. 포함관계는 유스케이스 분기가 필수적, 확장 관계는 유스케이스 분기가 선택적이다. 라는 말을 잘 이해하지 못했습니다.

4. 너무 낮은 레벨의 유스케이스에서 낮은 레벨에 대한 개념이 잘 이해가 가지 않습니다.

# **제 4장. 클래스 다이어그램**

|  |  |
| --- | --- |
| 배려의 세 가지 조건 | 마음을 움직이는 힘 -> **배려**  배려하려면 솔직해야 하고 상대방 관점에서 봐야 하고 노력하고 통찰력이 있어야 한다.  배려는 상대방을 위해서가 아닌 자기 자신을 위해서이다. |

## **4.1 클래스 다이어그램의 구성요소와 표현**

|  |  |
| --- | --- |
| 클래스 | **객체를** **추상화시킨** 것  동일한 유형 의 객체 -> 객체와 **상태와 행위가 유사한** 것 |
| 객체의 일반적인 특성 | -> 실제로 존재하는 구체적인 대상  -> **다른 객체와 구분되며 고유함**  -> 상태를 가지며 행위를 제공  -> 문제 영역(개발) 에서 의미 있는 존재 |
| 클래스 다이어그램의 목표 | 시스템 내에서 **모델링 되는 유형**을 보여주는 것  -> 클래스, 인터페이스, 데이터 유형, 컴포넌트 |
| 클래스 다이어그램 정의 | **객체 타입(클래스)를 정의**하는 것 (ER 다이어그램과 유사)  **소프트웨어를 개발하는 시스템에 핵심적인 클래스를 정의**함으로써 설계를 하는 것 -> 향후 설계는 구현을 위한 것  클래스와 그 관계로 구성되어 있는 것이 SW 시스템 |
| 클래스는 정적 | 클래스는 **정적**이다.  정적 -> 시간과 조건이 개입되지 않기 때문에 클래스는 본질적으로 **정적**이다.  -> **객체는 반대로** 그 자체가 **동적**인 개념 |
| 클래스 다이어그램 용도 | -> 프로그래머가 시스템을 바로 프로그램 코드로 바꿔줄 수 있는 설계도 작성  -> 클래스 다이어그램은 따라서 개발에 가장 중요한 다이어그램이다.  -> 다른 다이어그램을 이해하기 쉽게 도와주는 보조적인 도구로도 사용 |
| 클래스 다이어그램 작성 목적 | -> **시스템을 분석하고 이해하는데 매우 용이**  -> 실제 설계과정을 실행함으로 써 **구축 과정을 해준다**.  즉 정의해 놓은 데이터와 함수(속성과 메서드)를 상세히 정의  -> **논리적 관점**으로부터 **물리적인 관점**까지 일관된 형식으로 **시스템 분석, 설계 방식 제공**  **-> 분석, 설계에 차이를 메꿔주는데 좋다.** |
| 클래스 다이어그램 작성 시기 | 시스템의 **분석 단계와 설계 단계 중간쯤에서 반복해서 그린다**. 반복하면서 점점 상세화 함  여러 번 작성하는 이유 -> 시기마다 관점(분석, 설계 등등)이 변하기 때문에  분석단계 -> 상위 레벨설계 단계 -> 상세 설계 단계 |
| 클래스 다이어그램을 작성시 필요한 것 | **사용자의 요구사항 정의 -> 유스케이스 다이어그램**  **정확한 scope -> 정확한 범위가 정의되어 있어야 함** |
| 클래스의 구성 요소 | 클래스 **이름, 속성, 메서드** |
| 클래스 이름 | 구체적이고 명확한 이름 -> 고유한 이름을 사용해야 한다. |
| 속성(attribute) | 클래스의 구조적 특성에 이름을 붙인 것  -> 특성에 해당하는 인스턴스가 보유할 수 있는 값의 범위를 기술 |
| 메소드(method) | 오퍼레이션이라고도 함  -> 실제 행위를 말한다. 클래스의 행위적 특징 |
| 클래스 다이어그램의 구성 요소 | **things와 relationships**  things -> 클래스  relationships -> 연관, 일반화, 의존 |
| 클래스의 표기 | **단순형 표기** -> 표현의 **양식의 이름만 있는 것.**  **정규형 표기** -> 전체에 대한 **모든 값을 다 표현한다**.  **클래스 다이어그램을 반복해서 그리면서 단순형 표기에서 정규형 표기로 바꿔 나가게 된다**.    클래스가 너무 많으면 상위 버전에서는 단순형으로, 내려가면서 정규형 표기로 한다. |
| 객체와 클래스 사이의 관계 표현 | 클래스 -> 객체를 생성할 수 있는 구조와 정보를 갖는 틀  객체는 클래스의 인스턴스, 객체들의 추상화 개념 |
| **클래스의 생성 과정** | 난이도가 어렵다. (클래스 만들기)  **후보 클래스 도출 -> 부적절한 클래스 제거 -> 클래스에 대한 검토 -> 클래스의 추상화 수준과 표현** |

## **4.2 클래스 다이어그램 관계**

|  |  |
| --- | --- |
| 객체와 클래스 사이의 관계와 표현 | **매우 중요한 개념**  연관 관계는 의존 관계와 마찬가지로 객체를 생성하는 관계  -> 연관 관계는 멤버 변수로 참조하고 의존 관계는 메서드로 참조한다는 것이 다르다.  **[매우 중요]**     1. 영원히 유지하고 있으면 의존 관계 |
| 연관 관계 | 클래스 간에 서로 개념적으로 연결된 선으로 표현한다.    각 클래스에는 각각의 역할이 존재 |
| 연관 관계의 다중성 | **다중성** -> 팀은 하나지만 선수는 한 명이 있을 수도, 여러 명이 있을 수도 있다. |
| **집합 관계의 복합 관계** | **연관 관계 안에서** 좀더 강하게 만들어 준 것  -> 집합 관계와 복합 관계는 연관 관계에 포함된다.  -> 복합 관계는 집합 관계에 포함된다.    -> 집합: 독립적인 구성 요소로 연관됨, 생명주기가 독립적, 전체 객체가 삭제되어도 부분 객체가 남는다.  (차와 엔진, 바퀴 차체) -> 집합 관계  차와 엔진은 독립적이 가능  -> 복합: 영구적이며 강한 연관 관계로 구성됨, 전체가 없어지면 부분도 없어진다.  (엔진과 피스톤, 플러그, 카뷰레터) -> 복합 관계  엔진이 없으면 엔진 부품은 의미가 없다. |
| 일반화 관계 | 한 종류로 묶을 수 있는 관계 |
| 의존 관계 | **하나의 클래스가 또 다른 클래스를 사용하는 관계**  -> **하나가 없으면 다른 클래스가 존재하기 어려워진다**.  -> 다른 클래스가 클래스에게 영향을 준다. **서로서로 영향을 준다.**  **하나의 클래스에 있는 어떠한 대상이 다른 쪽에 영향을 준다**  **-> 의존 관계 (TV 와 리모컨)** |
| 실체화 관계 | **추상 클래스나 인터페이스를 상속받아서 하위 클래스에서 그 메소드를 실제로 구현하는 것**  추상 클래스: **아직 상위 레벨인데 완벽하게 애트리뷰트가 없는 경우, 메소드밖에 없는 경우**.  인터페이스: 상수와 추상 메소드밖에 없다. **메소드조차 없는 경우** |

## **4.3 클래스 다이어그램의 단계별 모델링**

|  |  |
| --- | --- |
| 단방향 연관 관계 | **한쪽 방향으로만 연관**되어 있다. |
| 양방향 연관 관계 | **양방향으로 연관**되어 있다. |

## **4.4 클래스 다이어그램을 효과적으로 작성 방법**

|  |  |
| --- | --- |
| 효과적인 작성 방법 | 1. **최대 15개 정도 최고 수준의 클래스 다이어그램을 작성** 2. **핵심 클래스 중에서 하나를 다이어그램의 중앙에 위치시키고 이를 지원하는 두 번째 수준의 다이어그램 작성** 3. **중요한 집단화가 존재하는 경우 별도의 다이어그램에 전체와 그 부분 간의 집단 관계를 표현** 4. **중요한 상속 계층이 존재하는 경우 별도의 다이어그램에 수퍼 클래스와 서브 클래스를 위치시킴** 5. **열 개 이상의 지원 클래스로 인해 복잡해진 경우 세번째 수준의 클래스 다이어그램 작성** |
| 클래스 다이어그램과 객체 다이어그램 | **클래스 다이어그램**  -> 시스템의 정적 설계 뷰를 모델링 하는데 사용  **객체 다이어그램**  -> **스냅샷**  -> **클래스 다이어그램의 실행시에 나타나는 하나의 예를 나타낸 것**  -> **클래스 대신 인스턴스를 보여준다.**  -> 클래스 다이어그램이 잘 그려졌는지 **검증용** |

## **4.5 Question**

1. 클래스 다이어그램 작성 목적에서 논리적 관점과 물리적 관점에 대해서 잘 이해가 되지 않습니다.

2. 클래스의 속성에서 인스턴스가 보유할 수 있는 값의 범위라는 설명이 잘 이해가 가지 않습니다.

3. 집합 관계에서 생명주기가 독립적이다 라는 말이 잘 이해가 가지 않습니다.

# **제 5장. 순차 다이어그램**

|  |  |
| --- | --- |
| 프로세스 중심으로 사고 | 체계적으로 일할 것  -> 모든 생각과 사고와 행동을 프로젝트 중심으로 체계적으로 하면 좋은 점이 매우 많다. |

## **5.1 순차 다이어그램의 구성요소와 표현**

|  |  |
| --- | --- |
| 순차 다이어그램이란 | 주어진 문제를 **해결**하는 모델  -> 클래스 양식에서 실제 필요한 오브젝트를 정의  액터가 어떠한 순서로 일하는가?  원하는 순서로 일하는 것을 실제로 그렇게 되도록 실현시켜주는 것 |
| 순차 다이어그램 특징 | 동적 모델링 기법을 제공한다.  순차 다이어그램은 시간적, 공간적 개념에서 상호 작용하는 다이어그램이다.  -> 순차 다이어그램은 **상호 작용하는 다이어그램의 일종이다**.  -> 오직 시간 개념이다.  -> **가능하면 위에서 아래로 흐르는 시간 개념의 다이어그램이다**. (항상 위가 아래보다 먼저 일어난 구조)  -> **시간 순서를 중시하고 처리를 위한 객체 간 메시지를 중요시함**  순차 다이어그램은 **유스케이스 다이어그램으로 그린다.**  -> 현재 시간을 설정  -> 큰 과정을 하나의 유스케이스로 본다.  -> 따라서 유스케이스가 매우 중요하다. |
| 순차 다이어그램의 작성 목적 | 유스케이스의 실현  -> 유스케이스 다이어그램에 있는 하나의 유스케이스를 정의한다.  -> 구현되기 전에 유스케이스를 그리고 순차를 그리고 구현을 한다.  -> 객체를 정의하고 메시지로 연결하면서 기능이 실현된다.  **동적으로 상호작용을 시간적 개념을 중시하여 모델링 한다**.  -> 정의된 객체가 어떤 일을 하는지 등등  객체의 오퍼레이션 속성을 상세히 정의한다.  -> 클래스 다이어그램을 적을 때 상세하지 못한 곳을 순차 다이어그램으로 상세화 해줄 수 있다.  -> 어떤 일을 어떻게 처리하고 어떻게 하라고 요청하는 것 어떠한 책임을 지는 것이 오퍼레이션이고 실제로 행위를 하는 것이 실제의 속성 값이다.  프로그래밍 사양을 정의한다.  -> 그대로 실현하는 단계 |
| 순차 다이어그램 작성 시기 | 유스케이스를 그리고 구현되는 상황 사이에서 순차 다이어그램을 사용한다.  -> 구현과 설계 단계에서 계속 피드백을 진행한다.  -> **유스케이스가 준비되어 있어야 하고 유스케이스 명세서가 작성 시에 매우 중요**하다. |
| 순차 다이어그램은 동적 | 순차 다이어그램은 **항상 액터로 시작**해야 한다. 액터에서 오브젝트를 주고받는다.  -> 반드시 액터가 있어야 하고 그 사이에 메시지가 있어야 한다, |
| 순차 다이어그램 구성 요소 | 순차 다이어그램 모델링에서 수직방향이 시간  **객체의 과정을 블록**이라고 한다.  이 개념이 **시간적 개념이냐 공간적 개념이냐에 따라서 순차 다이어그램, 통신 다이어그램을 구분**한다. |
| 순차 다이어그램의 표현 | 객체 위에서 아래로 뻗어 나가는 점선, 활성화되는 부분을 표시한다.  **점선이 생명선이고 박스가 활동하는 과정** |
| 분석 단계와 설계 단계 | 구조적 모델에 매우 영향을 받는다.  **클래스 중심으로 그릴 것인가 오브젝트 중심으로 그릴 것인 가** |
| 구성 요소 | 대상이 -> 액터, 객체,  이들 사이에서 주고받는 것이 -> 메시지  -> 액터와 오브젝트의 사이를 주고받는 것이 메시지  좀더 풍부하고 이해를 원활하게 하기 위해 사용하는 것  -> **생명선, Focus of control**  -> **생명선:** 액터나 객체의 시작과 끝을 표현한 것이 생명선  -> 생명선이 살아 있지만 실제로 시작과 끝을 표현해주는 것이 **활성화(Focus of control)** |
| 액터 | 순차 다이어그램을 **시작**하는 역할 |
| 객체 | 객체명: 클래스명 -> 이렇게 표기 |
| 메시지 | 객체끼리 전송되는 것  -> **생명선을 이동**한다. |

## **5.2 순차 다이어그램의 단계별 모델링**

|  |  |
| --- | --- |
| 모델링 및 프로세스 | **작성 대상 선정(명세서 정의) -> 액터를 위치시키기 -> 클래스(객체)를 위치시키기 -> 액터/객체간 메시지 정의하기-> 객체 추가로 정의하기** |
| 순차 다이어그램 모델링 | 작성 대상을 선정  -> 시간적으로 처리 과정이 많은 것을 명세서로 작성  유스케이스의 액터를 파악하여 다이어그램에 위치  -> 모든 시작은 액터로 시작되어야 한다.  유스케이스를 실현키 위해 참여할 클래스(객체)들을 정해 다이어그램에 위치  시간 순서를 감안하여 액터, 객체간 메시지를 정의  필요한 객체를 추가로 정의 |
| 작성시 주의사항 | **순차 다이어그램은 유스케이스 별로 하나씩 작성**  동일한 상호작용을 여러 순차 다이어그램에서 중복되게 작성 피할 것  중복을 최소화 시키기 위해서 **UI별 순차 다이어그램 작성**  **메시지의 흐름은 액터로부터 시작되게 작성**  클래스 다이어그램에 표기된 클래스명과 매핑 가능하도록 객체이름을 표시  가독성이 좋도록 **적당한 주석** 사용 |

**5.4 Question**

1. 객체 책임과 오퍼레이션, 속성의 정의가 잘 이해가 되지 않아 한번 더 설명해 주셨으면 합니다.
2. 객체표기예의 두번째에서 클래스가 아직 정의되지 않은 ‘홍길동’ 이라는 객체는 어떻게 형성된 것인가요?

# **제 6장. 통신 다이어그램**

|  |  |
| --- | --- |
| 상호 관계성 | 동일한 용어, 동일한 이해, 동일한 마인드를 가지는 것  -> 공통 툴을 사용하고, 공통 개념을 이해하고, 공통 지식을 가지는 사람이 될 것  -> 상대방을 배려하는 사람이 되자  -> 남이 나를 알아주지 못함을 탓하지 말고, 내가 남을 알지 못함을 걱정하라 |

## **6.1 통신 다이어그램 개요**

|  |  |
| --- | --- |
| 정의 | 두 객체에서 상호작용하는 방법 중 순차 다이어그램과 통신 다이어그램이 있다. |
| 통신 다이어그램 | 문제를 해결하기 위한 객체를 정의 -> 객체를 찾아내서 어떻게 움직이는지 **상호 관계에 따라 정의해준다**.  **하나의 유스케이스를 어떻게 구현해주느냐가 중요**  -> 기존의 순차는 순서에 관계  -> 통신 다이어그램은 상호 협력이 중요  -> **구조적인 측면을 중시하여 모델링 한다**. |
| 작성 목적 | **유스케이스 단위로 그린다**.  -> 통신 다이어그램도 그리는 단위는 유스케이스다.  -> 유스케이스 안에 있는 객체가 중심이 된다.  -> 객체가 메시지를 통해서 유스케이스를 실행  **객체간의 동적 상호작용을 구조적 측면을 중시하여 모델링**  -> 객체가 어떻게 상호작용 하는지를 정의  **객체를 좀더 상세하게 정의한다.**  -> 애트리뷰트와 메소드를 좀 더 상세히 한다  -> 책임이 더욱 상세하게 정의  **프로그래밍 사양을 정의**  -> 상세화된 통신 다이어그램은 곧바로 프로그래밍 될 수 있는 수준으로 프로그램 사양 정의가 가능 |
| 통신 다이어그램 작성 시기 | 유스케이스 다이어그램부터 구현 까지 작성 하는 시기중  -> **시퀸스 다이어그램과 동일하다**. -> 여러 번 피드백  분석 단계에서는 비즈니스 관점  설계 단계에서는 구현 관점  레벨별로 필요한 관점이 있듯이 통신다이어그램의 관점이 중요  **작성 전에 유스케이스 명세서가 필요** |
| 통신 다이어그램 | **객체 다이어그램의 확장**  객체 다이어그램은 클래스 다이어그램의 스크린 샷  통신 다이어그램은 기본 뼈대가 된다.  하나하나의 스냅샷을 연결 해서 하나의 영화로 만드는 것  객체 다이어그램에 메시지를 추가해준 것이 통신 다이어그램  순차 다이어그램과 통신 다이어그램은 바꿀 수 있다.  순차 -> 순서  통신 -> 구조 |

## **6.2 통신 다이어그램의 구성 요소와 표현**

|  |  |
| --- | --- |
| 구성 요소 | 통신 다이어그램은 액터, **객체**, **메시지**, 링크로 구성 |
| thing | 액터, 객체 |
| relationships | 링크, 메시지 |
| 객체(object) | **메시지를 송신 수신하는 주체**  -> 클래스 다이어그램에서 **일관성 있게 그대로 클래스 명을 가져와야 한다.** 이후 객체명을 표기한다. |
| 활성 객체 | 여러 개의 개체 중 흐름을 제어하는 객체 -> 두꺼운 사각형 |
| 동시성 | 시스템 내에서 두개 이상의 객체가 동시에 작동될 수 있다. |
| 링크 | 단순한 연결 통로  -> **양방향 연관 관계**  -> 화살표가 없는 실선으로 표시하고 이름 붙일 수 있음 |
| 메시지 | 객체에 적힌 설명, **반드시 번호를 붙여서 순서를 매겨야 한다.**  -> **여러 객체로 메시지 전송**  -> 메시지에는 \*가 붙은 대괄호 조건문을 붙여준다. |
| 통신, 순차 비교 | 순서가 중요한 개념을 구조적으로 바꾼 것이 통신 다이어그램  두 객체간의 관계가 중요하다.  -> 구조적인 측면이 중요하기 때문에 시스템 아키텍처가 통신 다이어그램을 많이 사용한다.  순차 다이어그램을 구조화 한 것이 통신 다이어그램 |

## **6.3 통신 다이어그램 모델링 (각 단계의 모델링 순서 암기)**

|  |  |
| --- | --- |
| **작성 단계** | **작성 대상 선정 -> 객체를 선정(액터) -> 객체를 다이어그램에 위치시키기 -> 객체간 링크 정의 -> 객체간 메시지 정의하기 -> 객체 추가로 정의하기 -> 객체간 링크 정의하기로 넘어가 이 과정 반복하기** |
| 작성 프로세스 | 작성 대상을 선정  -> 명세서를 보고 작성  필요한 객체와 액터를 선정  중요한 객체를 중앙으로 배치하여 객체들을 다이어그램에 배치  링크를 정의  메시지 정의 |

## **6.4 통신 다이어그램 작성 방법**

|  |  |
| --- | --- |
| 그리는 방법 | 유스케이스 명세서에서 모델링 하는 방법  순차 다이어그램이 있다면 순차 다이어그램에서 변환해나가는 방법 |

## **6.5 순차 다이어그램 단계별 모델링**

|  |  |
| --- | --- |
| 디지털 도어 록 시스템 표현 | 사용자, 스마트 카드, 도어 록 클래스 생성 이후  이 3가지를 이어 만든 객체 다이어그램 생성  이후 같은 방향에 여러 메시지가 있다면 메시지 화살표를 하나만 그리고 그 위에 각각의 메시지 이름을 모두 표시한다  전혀 다른 종류의 메시지는 따로 그려준다.  클래스 -> object -> 통신 -> 순차 |

## **6.6 통신 다이어그램 작성시 주의 사항**

|  |  |
| --- | --- |
| 작성시 주의사항 | 시나리오를 분명히 해야 함  -> 명확해야 한다. 기능과 순서가 명확해야 함  다이어그램은 1개당 하나의 유스케이스로 그려야 한다.  -> 단일기능이 가장 좋다.  긴 시나리오는 다이어그램을 분할해서 그린다.  메시지명 받는 측의 관점에서 붙인다.  모르는 오브젝트에 메시지를 보내지 않는다.  오브젝트의 생성과 소멸을 의식한다.  상호 작용에 참가하지 않는 오브젝트는 그리지 않는다.  오브젝트명을 붙임  파라미터도 추가한다.  상세함에 주의한다. |

## **6.7 Question**

1. 흐름을 제어하는 객체를 활성 객체라고 하셨는데 다른 객체에 비해서 활성 객체가 될 수 있는 조건이 무엇인가요?
2. 작성시 주의사항에서 통신 다이어그램을 그릴 때에 파라미터, 조건문을 어떻게 그리나요? 통신 다이어그램에서 무엇을 파라미터라고 지칭하는지 잘 이해가 가지 않습니다.
3. 통신 다이어그램 예시에서 객체에 밑줄의 의미는 무엇인가요?

# **제 7장. 활동 다이어그램**

|  |  |
| --- | --- |
| 역할과 책임 | 개발에서 성과 개선 세부 역할 등 각각의 규칙이 있는데 각 조직에 대한 규칙을 잘 지키고 어떠한 일을 하고 분야별 전문가가 되도록 노력하자. 회사를 들어가기 전에 필요 역량이 무엇인지 잘 파악하자. |

## **7.1 통신 다이어그램 개요**

|  |  |
| --- | --- |
| 활동 다이어그램 정의 | **처리 흐름을 순서에 입각하여 정의**한 모델 **플로우 차트와의 연관성** 있음. 어떠한 영역에서 **처리 로직, 조건, 경우의 수에 따라 흐름이 어떻게 바뀔 것**인가 잘 처리한다.  **로직에 조건이 있다** 보니 그를 처리하는 **분기 처리 절차 표현이 가능.**  -> **하나의 행위**에서 **다음 행위**로 순서가 바뀌면서 처리하는 과정을 활동 다이어그램이라고 한다.  소프트웨어 처리, 순서를 표현해주는 것을 말함 |
| 활동 다이어그램 작성 목적 | **비즈니스 프로세스 정의 -> 비즈니스 모델링.**  -> 우리가 가지고 있는 현재 프로세스가 무엇이고 어떻게 개선해서 To-Be로 프로세스 작성  -> To-Be를 찾아라(미래의 개선 모델)  **유스케이스 실현**  -> **유스케이스를 가지고 처리 흐름을 순서도로 상세히 기술하는 것**  **대상 상관없이 처리 순서 표현**  -> 로직 처리순서의 표현에 따라 활동 순서가 어떻게 바뀔 것인지  **프로그램 로직 정의**  -> 흐름에 따라서 명확하게 의사 결정된 것을 처리 로직으로 표현해줌으로서 사용자에게 설명하기에 편하다.  -> 로직의 모델화  -> 자연언어로 기술하는 것은 부적절 |
| 활동 다이어그램 작성 시기 | **업무 비즈니스 시점**  **유스케이스 정의서 작성 시점**  -> 명세서 정의할 때  **오퍼레이션 사양 정의 시점**  -> 클래스 오퍼레이션 사양  **처리 흐름이나 처리 절차를 명확히 정할 때**  -> 명확하게 경우의 수를 정의할 때 사용 |
| 활동 다이어그램 구성 요소 | **시작, 종료, 액티비티, 판단, 동기바, 전이, Swimlane**  릴레이션이 아닌 순서에 대해 바뀐다.  복잡하게 있는 다이어그램을 좀더 구분하고 역할을 명확하게 하기 위한 게 수영장 라인을(구획면) 정해주는 것  새로운 업무 부서를 만들 때 활동 다이어그램 활용 가능  시작과 끝점에서 모든 것이 전이한다.  여기서 행위 할 수 있는 건 액티비티  두개로 나가는 것 분기 한 개로 들어오는 건 합병  동시에 하는 동기에 대한 개념도 있다. |
| 액티비티 정의 | **행위(액션)나 작업(태스크) 하나의 처리 단위**  **가능한한 크기가 비슷하게 하는 것이 좋다**.  **내부적으로 구조를 가질 수 있는 단위**  **클래스 다이어그램의 행위**이다. |
| 시작점, 종료점 | 시작과 끝을 정의하는 개념 -> 항상 있어야 한다. |
| 판단 | 판단은 **분기가 일어나는 곳**  **분기** -> 조건 분기로 가서 2개 이상의 조건이 동시 성립 불가  **합병** -> 조건 분기에서 나누어진 이동을 1개로 합치는 것 |
| 동기바 | 병렬 처리절차가 시작되거나 모이는 곳(포크 같은 거)  시작은 **포크**, 모이는 곳은 **조인**  순서에 상관없이 전부 올 때까지 다 기다려야 한다.  다 끝나야 지만 다음 단계로 넘어간다.  1개에서 n개로 나뉘는 것 -> 포크  n개에서 1개로 모이는 것 -> 조인 |
| 전이(Transition) | 하나의 액티비티가 행위를 완료하면 다른 액티비티로 처리 순서를 옮겨준다.  하나의 액티비티에서 **여러 개의 전이도 가능** |
| Swimlanes(구획면) | 누가 할 일을 누가 관장할 지 구획을 나누어 주는 것  누가 무엇을 한다. 구역 내의 행위를 무엇을 한다라는 식으로 표현 가능 |
| 활동 및 전이 | **활동 상태 -> 처리와 실행**  전이 -> **활동의 처리 순서**를 바꿔준다. |
| 분기 | **조건에 따라서** 행위만 해주면 된다.  **활동 흐름이 2가지로 나뉘며 여러가지 활동 상태로 분기** |
| 동기화 막대 | **동시에 시작하고 동시에 끝난다는 뜻을 의미** |
| 신호 | 주고받는 것, 보내주고 받는 것 신호  어디로 보내주는 신호와 받는 신호가 있고  시간을 나타내는 신호 (시그널) (송신, 수신, 시간) |
| 구획면 | 구체적으로 역할을 나누어 줘서 영역을 표시해 상대가 이해하기 용이하도록 한다. |

## **7.2 활동 다이어그램 용도**

|  |  |
| --- | --- |
| 활동 다이어그램 | 주로 유스케이스 수준 또는 상위 수준에서 **비즈니스 프로세스 표현**  **유스케이스 하나에 대해서 안에 어떤 흐름을 가지는지** 잘 볼 수 있도록 해준다.  설계 단계에서는 **오퍼레이션**, 알고리즘이나 구체적인 로직 표현  순차, 통신 다이어그램에서는 나타내기 어려운 상황을 표현  **업무의 흐름 분석**, **화면** **흐름 제어, 업무 흐름 제어** |
| 활동 다이어그램 업무 흐름 분석 | 구획면, 분기, 동기화 막대  -> **업무 흐름을 표현할 때 가장 효과적인 다이어그램이 활동 다이어그램이다.** |
| 활동 다이어그램 화면 흐름 분석 | 실제 화면 흐름에서 나오는 것은 유스케이스 인터페이스임  클래스 다이어그램을 그릴 때 클래스의 메소드 부분이 활동 다이어그램의 **전이**(흐름)이다. |

## **7.3 활동 다이어그램 작성 순서**

|  |  |
| --- | --- |
| 활동 다이어그램 작성 순서 | 1. **작성 대상을 선정**   유스케이스를 정해서 유스케이스의 실제 명세서를 보고 작성   1. **활동 상태 추출** 2. **모델링** 3. 사전 조건 모델링 4. 유스케이스 종료와 사후 조건 확인 5. 선택 흐름 조건 작성 6. 동기화 조건 작성 7. Swimlanes 정의[필요시] 8. 시작과 끝 부분 확인 9. **전체 모델링 조합** |

## **7.4 활동 다이어그램 작성시 주의사항**

|  |  |
| --- | --- |
| 활동 다이어그램 작성시 주의 사항 | 1. **꼭 필요한 요소들만 표현**할 것 2. **추상화 수준에 맞는 상세화를 일관되게** 작성 3. **이해하기 좋은 단위로 나눠서 표현**   -> **액티비티 크기는 일정**하게  4. 목적을 전달할 수 있는 **정확한 명칭 부여**  5. **뒤에서부터 그린다.**  6. **분기와 합병, 포크와 조인을** **구별**하여 사용  7. 중요한 부분은 노트(주석), 색을 활용하여 **시각적 효과** |

## **7.5 Question**

1. 동기화 막대에서 동시에 시작한 작업이 동시에 끝나지는 않아도 되거나 혹은 그 반대의 상황도 가능한가요? 즉 포크만 있거나 조인만 있는 경우가 있나요?
2. 신호와 분기에 대한 구분이 잘 되지 않습니다. 신호는 단지 다른 액티비티에 정보를 제공하는 것인가요?
3. 선택 흐름을 다이어그램에 표시할 때 분기와 비교하여 조건을 활용한 이후에 어떻게 표시해야 할지 잘 모르겠습니다.
4. 액티비티가 내부구조를 가진다고 하였는데 어떤 방식으로 구성되는지 궁금합니다.
5. 판단의 정의에서 분기는 2개이상 조건이 동시 성립이 불가능하도 하였는데, 하나의 액티비티가 끝나고 2가지 활동이 생기는 경우는 분기가 아닌 건가요? 아니라면 어떻게 표시해야 하나요?

# **제 8장. 상태 다이어그램**

|  |  |
| --- | --- |
| 변화와 혁신 | 지금 있는 상태에서 다음 단위로 변할 때 더 좋게 변해야 한다. 높이 나는 새가 상승 기류를 탄다.  긍정적으로 생각하자. |

## **8.1 상태 다이어그램 표현과 용도**

|  |  |
| --- | --- |
| 상태 다이어그램 정의 | **라이프 사이클 동안 하나의 객체가 어떻게 바뀔 것인가 정의**  -> 상태 다이어그램  시스템을 처리하는 과정에서 하나의 대상물이 하나도 빠지지 않고 자기가 변화할 수 있는 모든 걸 추정하여 완전성을 끝까지 갖추기 위해서 -> 자동차 같은 복잡한 거 할 때 사용  **상태 변화가 많을 때 사용한다**.  우리가 가지고 있는 오브젝트가 어떠한 이벤트에 대해서 상태가 바뀌고 액션을 하는 것  -> 객체가 이벤트를 통해 바뀌고 액션을 하고 이벤트가 바뀌고 상태가 바뀐다. |
| 상태 다이어그램 작성 목적 | 객체의 상태 변화를 상세히 분석 복잡한 상태로 변화하면서 존재  **이벤트에 따라 상태를 움직인다**. **이벤트가 가장 중요**하다  이벤트와 상태가 합쳐져 속성과 오퍼레이션으로 바뀜  속성 -> 상태에 대한 값  오퍼레이션 -> 이벤트  상태가 속성의 벨류 이벤트가 오퍼레이션 |
| 상태 다이어그램을 작성하는 시기 | 보통 **클래스가 정의된 후**  **최소한 클래스 다이어그램과 순차 다이어그램이 있어야함** |

## **8.2 상태 다이어그램 구성 요소**

|  |  |
| --- | --- |
| 구성 요소 | things: **상태, 시작 상태, 종료 상태**  Relationships: **전이** |
| 상태의 정의 및 표기법 | 2가지  기본형 표기: 이름만 적는 것  상세형 표기: 상세형 안에 동작까지 표기한 것 상태가 특정한 값에 의해 변함 |
| 동작의 종류 | **진입 동작-, 탈출 동작, 활동, 내부 전이, 지연 사건**  내부 전이: 안에서 머물면서 다른 사건을 처리하는 것  지연 사건: 빠져나갈 때 기다리는 것 |
| 전이 | **상태와 상태 사이에 변화를 주는 것** -> **관계**  -> 이러한 원래 상태에서 어떠한 이벤트가 나와서 이 조건이 참이라면 이 동작을 함으로써 타겟에 대한 상태로 바뀌는 개념  전이 위에 올라오는 것  촉발 조건  전이 조건  동작  -> 이 3가지가 충족되면 다음 단계로 넘어간다. |
| 전이 조건 | 순차적 하위 상태: 순서적인 개념  동시적 하위 상태: 복합 상태 |
| 이력 상태(history) | 결과 값을 가지고 올라간다.  깊은 이력 상태  얕은 이력 상태 |

## **8.3 상태 다이어그램 표현과 용도**

|  |  |
| --- | --- |
| 상태 | 객체가 액션 할 수 있는 가지고 있는 조건 중 하나  -> 어떠한 이벤트에 어떠한 조건에 따라 |
| 상태가 중요한 것 | 객체가 가지고 있는 모든 가능한 경우를 상태로 파악해야 한다.  하나하나의 상태가 임의의 순간에 나타난 값 |
| 시작상태와 종료 상태 | 상태는 라이프사이클 개념이 있어서 시작상태와 종료 상태 표시를 반드시 해줘야 한다. |
| 표현 과정 | 먼저 things를 찾아라.  이후 상태를 찾아라 |
| 이벤트와 정의 | 사건을 가지고 이벤트라고 한다.  전달에서 바뀌는 것이 전이  전이 위에 어떠한 이벤트가 있는 것  이벤트가 있어야 전이가 된다. |
| 확장된 표기법 | **이름만 있다가 활동까지 들어가는 것 -> 확장**  이벤트와 전이의 확장은 전이에 여러 개의 조건을 많이 써서 제한 조건을 많이 둬 여러 개의 상황을 다양성 있고 경우의 수를 높이는 것 |
| 중첩된 상태의 개념 | 높고 낮은 2가지 관점  슈퍼 상태: **가장 복잡하면서 높은 상태 아주 일반적인 문제에 초점**  서브 상태: 확장된 **슈퍼상태 내부에 있음**  **내려갈수록 더욱 구체화된다.** |
| 제어와 분리 | 하나가 여러 개를 하는 다수의 작업을 진행한다.  다수의 작업을 할 때  동기화 막대에 의해 수정 |

## **8.4 상태 다이어그램 표현과 용도**

|  |  |
| --- | --- |
| 상태 다이어그램 모델링 | 1. 작성 대상 객체를 선정 2. 객체가 가지는 상태를 정의하여 나열 3. 상태와 상태간 전이를 정의 4. 전이와 상태의 상세한 부분을 정의   상태를 먼저 찾는 게 좋다.  그에 따른 행위와 조건을 찾아라 |
| 작성시 주의사항 | 1. **객체 하나에 대한 상태 변화를 표현**   -> 객체의 범위가 명확하게 되기 위함  -> 두개가 되면 다른 상태가 바뀌기 때문에 변화가 많은(전이가 많은) 한 개의 다이어그램을 그릴 것   1. 블랙홀 상태를 주의   -> **들어가는 전이가 있는데 나오는 전이가 없는 것**  -> end로 나와 줘야 되는데 빠져나오지 못하는 경우  -> 무한 루프를 통해 종료 조건이 없는 것  -> 큰 오류이기 때문에 검증 필요   1. **클래스/시퀀스 다이어그램과의 일관성에 유의**   -> 상태 다이어그램은 클래스나 순차 다이어그램에 썼던 오퍼레이션 메소드나 속성값을 **일관성 있게 그대로 가줘야 하지 새로운 것을 만들면 안된다.**  -> 전체에 대해서 신뢰성이 떨어지기 때문임  -> 반드시 메소드나 속성값을 그대로 가져와 사용 |

## **8.4 Question**

1. 내부 전이에서 현 상태에서 처리할 수 있는 이벤트를 처리한다고 하였는데 내부에서 처리가 가능하다면 활동과는 어떤 차이가 있는 건가요?
2. 지연사건에 발생한 것처럼 이라는 설명이 잘 이해가 가지 않습니다.
3. 어떠한 경우에 복합 상태가 일어나나요?
4. 슈퍼 상태와 서브 상태에서 중첩을 여러 번 할 수 있나요?
5. 슈퍼 상태에서 높은 관점과 낮은 관점에 대한 차이를 잘 이해하지 못했습니다. 개념이 더욱 상세히 내려가는 것을 말하는 것인 가요?

# **제 9장. 컴포넌트 다이어그램**

|  |  |
| --- | --- |
| 독립적, 독자적 | 레고 하나하나는 독립적으로 되어 있다. 하나하나가 모여서 사물이 되고 개념이 된다. 전체의 많은 개념과 내용 하나하나는 작은 습관 하나하나가 모이는 것이다. 습관은 배신하지 않는다. 스스로 자립하고 강해질 것 |

## **9.1 컴포넌트 다이어그램 표현과 용도**

|  |  |
| --- | --- |
| 컴포넌트란? | 부품.  **조립을 할 수 있는 프로그램의 최소 단위**  -> **자체적으로 실행 가능**하다.  정의: 컴포넌트를 어떻게 정의하고 그 사이가 어떻게 관계 되어있는지  시스템의 모듈화 된 유닛, 최소 단위  인터페이스를 통해서 제어하고 인터페이스 뒤에 숨는다.  언제나 **교체가 가능**해서 **기존의 개념을 새로운 기능으로 변경이 가능** |
| 컴포넌트 개념 | **소프트웨어:** 시스템을 다른데 사용할 수 있는, **재사용할 수 있는 최소 단위**  즉 시스템을 **업그레이드할** 수 있는, **독립적으로 실행될 수 있는 단위**  UML에서 말하는 컴포넌트: 실제로 구현해주는 요소 객체에 따라 기능과 데이터가 합쳐져 있는 단위(객체)  CBD 관점에서 말하는 컴포넌트: 독립적으로 개발, 배포, 조립이 가능한 시스템의 구성 단위(라이브러리, 펑션 등) |
| 컴포넌트 다이어그램 작성 목적 | 컴포넌트를 정의  컴포넌트간 의존관계가 어떻게 되는가?  실행 모듈(이것도 컴포넌트)이나 소스코드, DB등의 상호작용하는 것을 찾아낸다.  이러한 컴포넌트다 다 같은 게 아니고 컴포넌트마다 양식이나 구조가 프로그램마다 조금씩 달라진다.  컴포넌트는 사전에 미리 정해진다. 정적인 상호작용을 정의  모든 상호작용을 정의 |
| 컴포넌트 다이어그램 작성 시기 | **설계 단계의 마지막 단계**  클래스 개념으로 **클래스가 완전히 완료된 후** |
| 컴포넌트 다이어그램 구성 요소 | **things와 관계가 있음**  things: **컴포넌트**, **인터페이스**  Relationships: **의존 관계, 연관 관계**  컴포넌트: **논리적 요소**들을 **물리적으로 실행 가능한 것을 묶어 놓은 것**  인터페이스: 실제 **컴포넌트 중**에서 **메소드(오퍼레이션)만 모아 놓은 것**  **의존 관계:** **컴포넌트**와 **컴포넌트** 사이의 관계  **실체화 관계**: **컴포넌트와** **인터페이스** 간의 관계 실제 값이 없이 **행위만 함** |
| 컴포넌트 | 보이는 것은 캡슐화 되어 인터페이스를 통해 보여 진다.  실행 모듈이 대부분이다.  분석, 설계 산출물도 포함 |
| 인터페이스 | 외부로 보이는 얼굴 속성은 약하고 오퍼레이션만 강한 것이 인터페이스 |
| 컴포넌트 다이어그램 | 컴포넌트와 인터페이스를 어떠한 관계로 의존관계나 실체화 관계로 묶어주는 것  반드시 탭이 달린 사각형에 이름이 있어야 함  패키지 안에 콜론 + 콜론  버전을 표시하는 꼬리표 값  패키지의 내부 오퍼레이션도 보여줄 수 있다. |
| 컴포넌트와 클래스의 **공통점** | **이름 있음**  **인터페이스** 실현 가능  의존성 일반화, 연관 관계  **인스턴스 받는다** -> 위에서 **상속받는 걸 인스턴스**라고 함  -> 클래스에서 **어떤 특정 오브젝트로 찍어내는 것도 인스턴스**라고 할 수 있음  상호작용한다. |
| 컴포넌트와 클래스의 **차이점** | **클래스는 단순히 논리적인 양식**  컴포넌트는 이 **양식의 실제 어떠한 값과 어떠한 메소드가 포함되어 실행하는 것**  컴포넌트는 **컴퓨터에 의해 실행되지만 클래스는 서류**이다.  **컴포넌트는 논리적 요소를 물리적으로 패키지화 한 것**  클래스는 속성과 오퍼레이션을 직접 가질 수 있지만, 컴포넌트는 오직 외부에 정의된 **인터페이스를 통해서만 오퍼레이션을 갖는다.** |
| 인터페이스 정의 | 클래스의 일종  외부에 공개할 목적으로 쓰인다.  인터페이스의 구현은 클래스나 컴포넌트에서 한다. |
| 의존 관계 정의 | 하나의 대상이 다른 쪽에 영향을 받는 것  대부분 동적 |
| 실체화 관계 정의 | 실제 정의하는 사물과 이를 구현하는 사물 간에 **표현하는 관계**  인터페이스(정의) - 컴포넌트(구현),  유스케이스(정의) - 컬레버레이션(구현)  인터페이스(정의) - 클래스(구현) |

## **9.1 컴포넌트 다이어그램 작성 순서**

|  |  |
| --- | --- |
| 컴포넌트 다이어그램 작성 순서 | 1. **컴포넌트 대상을 정의** 2. **컴포넌트를 식별**   -> 컴포넌트가 무엇인지, 실행할 수 있는 모듈이 무엇인지 찾는다.   1. **컴포넌트를 배치**하고 필요시 **인터페이스를 붙임** 2. **의존 실체화 관계를 정의**   -> 실행하는 모듈인가 실행하지 않는 모듈인가 구분 |
| 작성시 주의 사항 | 1. 컴포넌트는 응집도는 높고 결합도는 낮은 단위로 정의   -> 어떠한 컴포넌트를 찾았는데 외부와 연결이 많다면 묶어줄 것   1. 컴포넌트 크기의 일관성 고려   -> 결합도가 낮더라도 일정하게 나누어 주는 것이 좋다.   1. 추상화 수준에 맞는 상세성을 일관되게 제공   -> 동일한 레벨에서 일관되도록 해줄 것   1. 목적을 전달할 수 있는 명칭을 부여 |

## **9.3 Question**

1. 컴포넌트 다이어그램에서 컬레버레이션은 무엇을 뜻하는 건가요?
2. 컴포넌트 다이어그램 작성 순서 4번에서 실행하는 모듈과 실행하지 않는 모듈에 대해서 의존 관계와 실체화 관계는 어떻게 매치되는 것인가요?
3. 클래스 기반의 컴포넌트와 클래스기반이 아닌 컴포넌트는 어떤 차이가 있나요?
4. 작성시 주의 사항에서 추상화 레벨이 같을 때와 다를 때의 차이가 무엇인가요? 그리고 추상화 레벨이 다를 때 왜 의미를 파악하기 힘든 지 잘 이해가 가지 않습니다.
5. 컴포넌트에 인터페이스를 의존 관계로 연결하는 것과, 인터페이스를 실체화하는 것에는 어떤 차이가 있나요?
6. 인터페이스에 제공 인터페이스와 필수 인터페이스는 어떤 차이가 있나요?

# **제 10장. 배치 다이어그램**

|  |  |
| --- | --- |
| 정리 정돈 | 어떠한 물건을 보려고 위치를 정리  꼭 필요한 것만 나타내서 필요할 때 잘 찾을 수 있도록  정리에는 삭제가 포함  정돈에는 삭제 없이 위치를 재조정  정리한 후 정돈 |

## **10.1 배치 다이어그램 표현과 용도**

|  |  |
| --- | --- |
| 배치 다이어그램 정의 | 분산된 장치 안에 컴포넌트를 **어떻게 잘 배치를** 해서 소프트웨어가 원하는 비기는 요구사항을 만족시키도록 하는 것  컴포넌트 다이어그램의 확장이다.  **노드와 노드를 간의 관계**  네트워크, 하드웨어 또는 소프트웨어들을 실행파일 수준의 컴포넌트들과 함께 표현 |
| 배치 다이어그램 작성 목적 | **소프트웨어 시스템을 실행하기 위한 하드웨어 자원들을 정의**  **그들의 관계를 어떻게 해줄 것인지**  시스템의 성능을 효율적으로, 비기능 요구사항을 올리는 것  물리적 장비들을 잘 연결해서 시스템의 성능을 올리는 것 |
| 배치 다이어그램 작성 시기 | **시스템 설계 단계의 마지막**  **소프트웨어 컴포넌트가 정의되고 하드웨어 사양도 확정된 이후에 작성한다.** |
| 구성 요소 | things: 노드, 컴포넌트  relationship: 연결(서버 - 서버), 의존(서버 - 컴포넌트) |
| 노드 | 가지고 있는 **하드웨어 자원**  즉 **소프트웨어를 동작 시켜서 주는 것**  + 외부 장치, 다양한 장비들이 노드로 정의될 수 있음  **노드는 처리 능력을 가진 장치**  노드 이름  노드 특성  실행 컴포넌트 |
| 연결 | 커넥션 – **물리적인 연결을 의미한다.**  **서버와 서버를 연결**  실선으로 표시  노드들의 **통신 방식을 표현**  **연결은 클래스와 마찬가지로 다중성을 가짐**  1:1, 1:n, n:n등 |

## **10.2 배치 다이어그램 작성 순서**

|  |  |
| --- | --- |
| 작성 순서 | 1. 노드를 식별하여 정의 2. 컴포넌트 식별 3. 노드간 구성관계를 정의 4. 노드에 컴포넌트를 배치 |

## **10.2 배치 다이어그램 작성 시 주의사항**

|  |  |
| --- | --- |
| 주의 사항 | 1. 목적을 전달할 수 있는 **명확한 의미의 명칭 부여** 2. 우리가 쓰고 있는 **하드웨어를 추상적으로 제공**해주어야 함 3. **모델을 만드는 목적을 어떤 목적으로 만들었는지 목적에 따라서 필요한 부분 까지만 분해** |

## **10.4 Question**

1. 배치 다이어그램의 작성 단위는 컴포넌트인가요? -> 노드도 things이다
2. 하드웨어의 자원을 배치할 때에 어떻게 성능과 효율에 영향을 주는지 잘 모르겠습니다.
3. 구성 요소의 연결은 실제로 네트워크 연결을 의미하는 것인가요? -> 노드의 연결은 네트워크 연결이 맞음, 컴포넌트 연결은 디펜던시이다.(의존)
4. 구성 요소 밖에 떨어져 있는 컴포넌트도 있을까요? 예를 들어 재고 조회 어플리케이션에 client 안에 device가 있는데 client 밖에 혼자 떨어져서 컴포넌트 혼자 존재할 수도 있을까요? -> 그런 건 없다. 책이 잘못 그린 것.
5. 하드웨어를 여러 개의 서버에서 분산되어 있는 것과 하드웨어를 추상적으로 작성해줘야 하는 이유의 상관관계를 잘 모르겠습니다.

모든 컴포넌트들은 의존이 되어있어야 프로그램이 돌아간다.

# **제 11장. 패키지 다이어그램**

|  |  |
| --- | --- |
| 패키지 | 하나하나 개별적인 것을 꾸러미로 만드는 것 -> 이것이 바로 컴포넌트.  소프트웨어를 어떻게 종합하여 프로그램을 만드는가?  행복한 사람은 내가 가진 것을 즐기고  불행한 사람은 남이 가진 것을 부러워한다.  내가 만족하지 않으면 절 때 행복할 수 없다.  내가 가진 것에 만족해야 행복해질 수 있다.  지금까지 한 작은 것이라도 내가 가진 능력이라 생각하고 가꾸어 나가자. |

## **11.1 패키지 다이어그램 표현과 용도**

|  |  |
| --- | --- |
| 패키지 다이어그램의 정의 | **모델의 요소,** 패키지 서버 소프트웨어 같은 것을 어떻게 묶어 줄 것인가? -> **계층적 구조를 제공해준다**.  패키지를 삽입하고 확장해서 모두 컴포넌트, 모델 요소들을 요소간 독립성을 부여  **클래스와 같은 여러 모델 요소들(컴포넌트)를 패키지로 구성하고 패키지 사이 관계를 표현** |
| 패키지 다이어그램 작성 목적 | 클래스 혹은 컴포넌트가 특정 서브 시스템을 구성하는 요소로 존재 -> **비즈니스 목적** |
| 구성 요소 | things -> 패키지  관계 -> relation(일반화, 의존) |
| 패키지 | **추상 개념이다.**  도서관에 책의 카테고리의 경계 값  모델링에 대한 단위로 지금까지 배운 모델링에 대한 요소들을 모두 그룹으로 묶을 수 있다.  패키지의 명칭은 문자열의 명칭을 가짐  클래스와 마찬가지로 꼬리표가 있다.  완벽한 패키지 이름 -> 패키지 이름::패키지 요소 |
| 패키지(구성 요소) | 요소들을 그룹으로 묶어주는 것 |
| 패키지 확장 표현법 | **패키지 안에 패키지 만들기 가능**  **패키지 안에 클래스 만들기 가능**  **패키지 안에 여러가지 이름 가능**  **클래스만 표현하는 방법**  **패키지에 경로를 표현**  **패키지 안에 패키지 포함 관계도 표현 가능** |
| 패키지 종류 | <<>facade> -> **다른 페이지에 보여주기만 하는 것**  하나의 인터페이스를 제공 뷰  <<framework>> -> 패턴을 제공**패키지 애플리케이션 개발**  <<stub>> -> 대리자 역할 **내려오는 모델이 스텁** **올라오는 모델이 드라이브**  <<subsystem>> -> 전체 시스템의 **독립된 일부분을** 나타내는 패키지  <<system>> -> 모델링 하려는 **전체 시스템을** 나타내는 패키지 |
| 가시성 | 어떠한 정보를 다른 그룹에 보여줄 것인가?  **공용(public) -> + 전체 공개**  **보호(protected) -> # 그룹원들끼리 만 공유**  **전용(private) -> - 공유 안함** |
| 일반화 관계 | **상위에 올라가는 개념**  클래스들 간의 일반화와 유사하다.  다른 쪽에서 다른 쪽을 사용하게 해준다. |
| 의존 관계 | **한쪽에 있는 것을 다른 한쪽에 이용하고자 하는 것**  한 쪽에서 export 다른 한쪽에서 import  **<<import>> -> 패키지에 있는 내부 참조를 허용해주는 것**  **상대방에 있는 패키지를 나에게 가져와서 추가**  **-> 나 한테 가져와서 실행하는 것**  **<<access>> -> 서버 통신을 하지만 실제 시행 또한 클래스를 가져오지 않음 -> 내가 가서 실행하는 것**  남과 연결만 해주는 것 |
| 의존 관계 | import -> 내가 있는 것을 허용하여 쓰면 된다.  -> **내가 하는 것을 내 쪽에서 오라**  export -> **남들이 써도 되지만 나에게 와서 만 써라**  -> **기존의 public 개념**  -> 허용해주니까 와서 써라  export후에 import를 할 수 있다. |
| 중첩된 패키지 | 패키지안에 패키지 포함 가능 |
| 클래스 다이어그램을 포함한 패키지 | 패키지 안에 클래스 다이어그램 포함 가능 |
| 패키지 합병 | **다른 패키지를 묶어 줄 수 있음**  **중첩되는 기능들을 하나로 묶어주어서 편의성 증진**  **기존의 개념에서 새로운 패키지로 변형**  **중복성을 방지해준다.** |

## **11.2 패키지 다이어그램 작성 순서**

|  |  |
| --- | --- |
| 작성 순서 | 1. **개념적으로 또는 의미적으로 가까운 요소들의 집합을 파악** 2. **Package에 대해 외부의 접근을 허용하는 요소 파악** 3. **외부 Package에 의존하는 Package들을 명시적으로**   **Import Dependency를 표현**   1. **Package가 Family를 이룰 경우 병합관계 표현.** |

## **11.3 패키지 다이어그램 작성시 주의사항**

|  |  |
| --- | --- |
| 주의 사항 | - **묶는 것은 좋으나 사이에 관계가 너무 많다면 좋지 않다.**  - 모든 시스템은 **응집도는 높고 결합도는 낮아야 하기 때문**에 묶을 때도 응집도와 결합도를 고려하자  - **결합도는 import할지 말지가 문제이다.**  **- 가능하면 import를 적게 하는 것이 좋다.**  - **패키지 작성 시에는 이름과 아이콘 쉽게**  - **뜻과 필요한 요소만 표현**  - **다른 사람들을 위해 절대 경로와 꼬리 표까지 하면 이해하기에 용이** |

## **11.4 Question**

1. stub이 다른 패키지의 공용 내용물에 대한 대리자 역할을 수행한다고 하였는데 다른 패키지에서 데이터를 복사해 오는 것을 뜻하나요?

2. <<subsystem>>에서 시스템의 일부분을 따로 표현하는 것의 이점은 무엇이 있나요?

3. import와 export에 대한 개념을 아직 잘 모르겠습니다. 간단하게 import는 데이터를 받아오는 것이고 export는 전달해주는 것인가요?

4. 패키지 안에 클래스 다이어그램을 포함시키는 것은 어떤 이점이 있나요?

5. import와 결합도가 어떠한 상관 관계가 있나요?

소프트웨어 아키텍처의 5개 뷰

어떤 업자가 많이 쓰는지 확인할 것

**설계 뷰**   **프로세스 뷰**

-> 정적: 클래스, 객체 다이어그램 -> 정적: 클래스, 활성 클래스 다이어그램

-> 동적: 상태, 순차, 통신, 활동 다이어그램 -> 동적: 상태, 통신, 활동 다이어그램

**유스케이스 뷰**

-> 정적: 유스케이스 다이어그램

-> 동적: 상태, 순차, 통신, 활동 다이어그램

**배치 뷰**  **구현 뷰**

-> 정적: 배치 다이어그램 -> 정적: 컴포넌트 다이어그램

-> 동적: 상태, 순차, 통신, 활동 다이어그램 -> 동적: 상태, 순차, 통신, 활동 다이어그램