동아설계도

목차

'동아 설계도' 의 필요성

- **Programing의 목적 => 문제해결**
  1) 인간의 삶을 전보다 더 **편리한 환경**으로, **만들기 위함**
  2) **인간**이 **명확하게 정의**하면 정의할수록 **컴퓨터**는 일련의 **작업**을 더 **원활하게 수행** 가능

★★ **Programming의 목적**은 **컴퓨터**가 **사람**에게 **'최대한'**의 **'서비스'**를 **제공**하는 것!

‘동아 설계도’ 의 필요성

❑ **코드 실행**을 위한 **근거가 부실**

1) z = x + y의 의미는 'z 는 x 더하기 y의 값을 가진다’이다.

-> 코드를 작성한 사람 외 사람들은 **왜** z 가 x+y 인지 모른다.

따라서 우리는 명확한 근거를 제시해 주어야만 한다.

2) 하지만 **z = find value such that ..구문**으로 **근거**를 **명확**하게 **제시**한다면, **컴퓨터가 알아서 처리**해줄 것이다.

즉, **사람들이 z = x + y 라 식을 적을 필요가 없어진다.**

‘동아 설계도’ 의 필요성

❑ 원본의 존재 이유

Ex) Fibo(n) = nth Fibonacci Number 를 구현하시오.

1) C언어 - Use Recursion

```
Fibo(1) = 1, Fibo(2) = 1

Fibo(x+2) = Fibo(x+1) + Fibo(x)
```

이는 **영속성이 존재**하지 않는 코드이다.

Fibo(n)을 사용하고 싶을 때 한번 짜고 버리는 코드….(?)

‘동아 설계도’ 의 필요성

❑ 원본의 존재 이유

```
Fibo(1,1) , Fibo(2,1)

Fibo(x+2, y+z) if Fibo(x+1, z) and Fibo(x,y)
```

=> 하지만 이 코드는 눈으로 보고 한 번에 이해하기 어렵다.

따라서 이를 **그림으로 쉽게 나타낸 것**이 우리가 제시하는 **‘동아설계도’** 이다.

‘동아 설계도’ 규칙

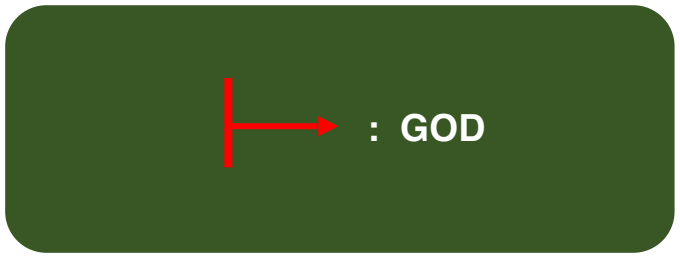
❑ 의의

- 각 설계도를 보면, **어떤 서비스가 어떤 객체에게 전달**되는지 **명확**하게 알 수 있다.

1. 헤더파일(GOD)의 존재

➢ 지구에 존재하는 발견과 발명의 모든 것이 담겨있다.(문명의 이기(利器) 포함)

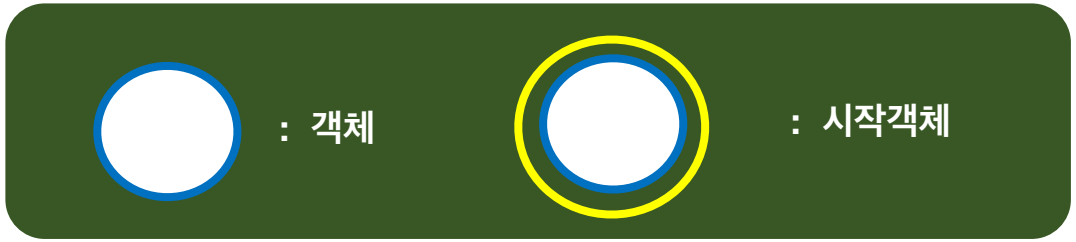
➢ 출처를 모르는 **능력은 전부 외부(GOD)**에서 지원한다



‘동아 설계도’ 규칙

2. 서비스를 최초로 시작하는 객체

- 서비스를 최초로 **시작하는 객체가 행동**을 취하지 않으면, 모든 객체들은 **좀비상태**에 존재한다.
- **동그라미 두 개 = 서비스를 최종적으로 제공받는 객체**
- **동그라미 한 개 = 그 외 객체**



3. 고유의 능력(서비스)이 존재

- 각 객체들은 **자신들이 주체적으로 생각하는 고유의 능력(서비스)이 존재한다.**
- 고유의 능력 = 까만 점으로 표현

‘동아 설계도’ 규칙

4. 실제 서비스의 흐름

➢ 실제 **서비스의 흐름은 점에서 점으로** 흘러간다. 만약 그 객체가 서비스를 처리할 수 없다면 **서비스는 처리(연결)되지** 않는다.

➢ 서비스의 흐름 = 까만 점끼리 화살표로 연결하여 표현

5. 예외 처리

➢ 각 객체는 자신이 **가지고 있지 않은 서비스를 요청하거나 제공할 수 없다**.

➢ 객체가 처리할 수 없는 서비스를 요청할 시 모두 거부

➢ **오류 발생!**

➢ **최초 설계 시 명확한 명세가 필요**

선택 OR (Choice OR)

❑ 서비스를 제공하는 쪽에서 선택

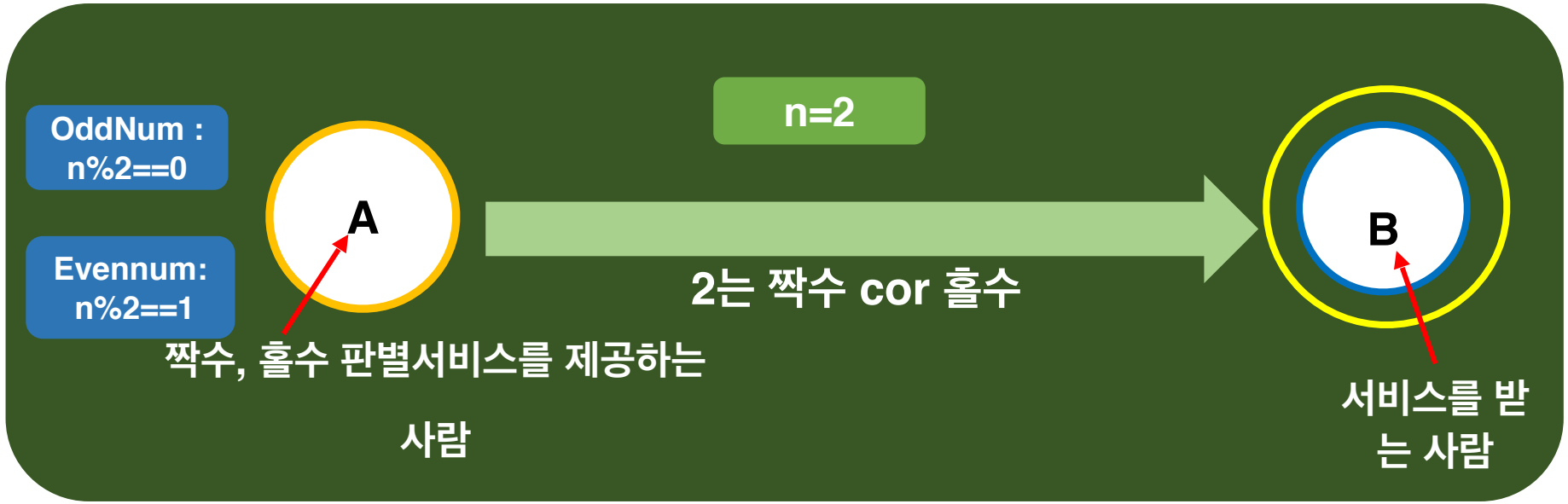
둘 중 하나를 선택하는데, 내가 아닌 **상대가 선택하는 것**이다.

=> A cor B : A, B를 선택하게 되지만, 상대가 선택해서 나에게 제공해주는 것이다.

Ex) 신호등에서의 빨간 불 cor 초록 불 cor 노란 불

신호등의 불빛색은 **신호등이 선택**한다. 내가 선택하는 것이 아니다.

선택 OR (Choice OR)



1. B는 A에게 “2는 뭐야?”라고 질문한다.
2. A는 B에게 2라는 값을 받아, 자신이 생각했을 때, 2는 짝수라고 결론을 내린다.
3. 마지막으로 A는 B에게 ”2는 짝수라고 답한다.

선택 AND (Chice AND)

❑ 서비스를 받는 쪽에서 선택

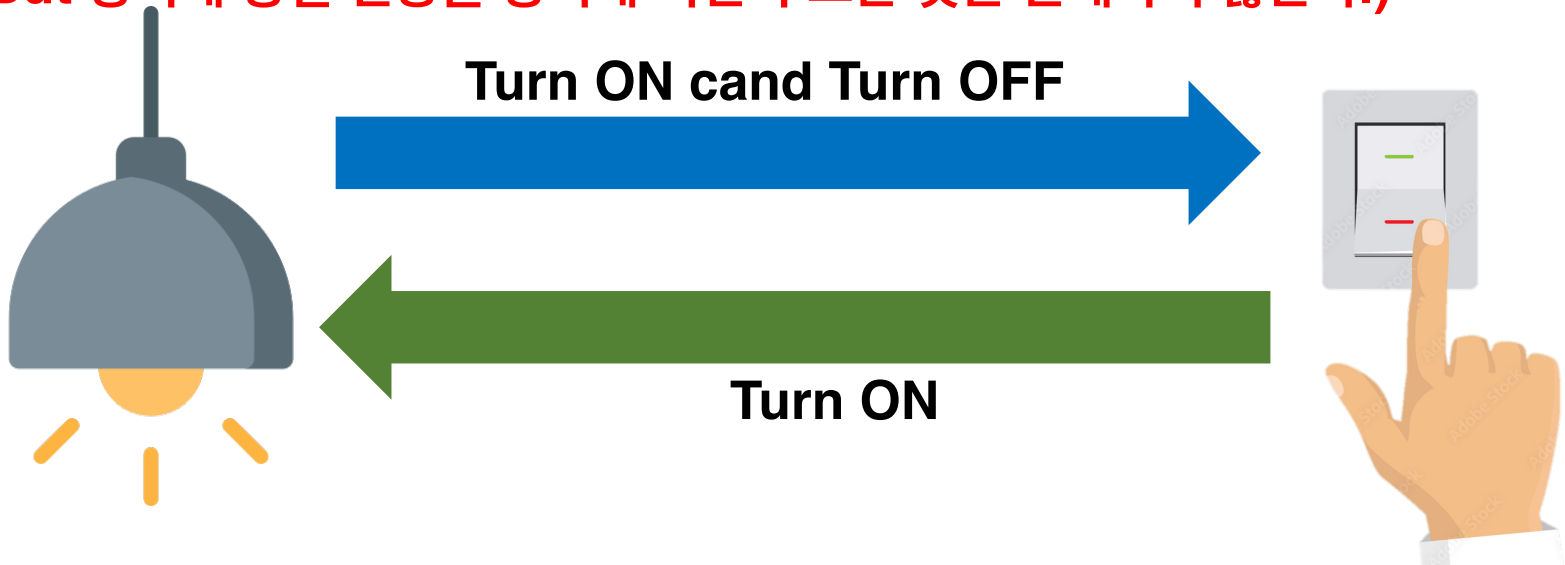
둘 중 하나를 선택하는데, **상대가 아닌 내가 선택**하는 것이다.

=> A cand B : A,B를 선택할 때, 내가 선택하는 것이다.

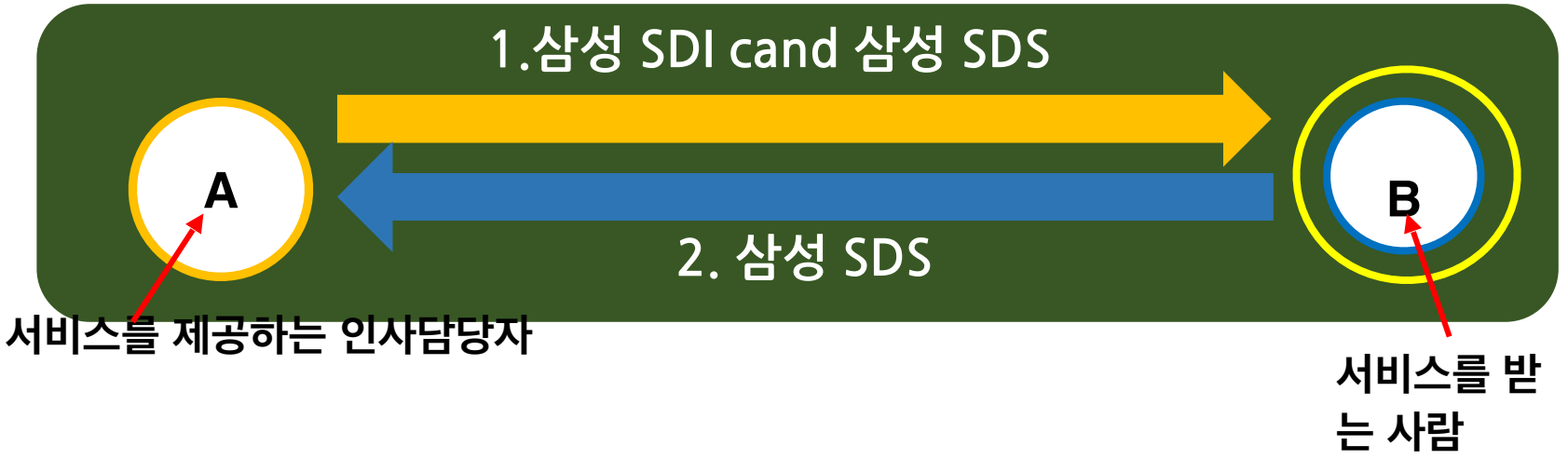
Ex) 전등을 켠다 cand 전등을 끈다.

내가 전등을 켜고 싶으면 켜고, 끄고 싶으면 끈다.

(But 동시에 동일 전등을 동시에 켜면서 끄는 것은 존재하지 않는다.)



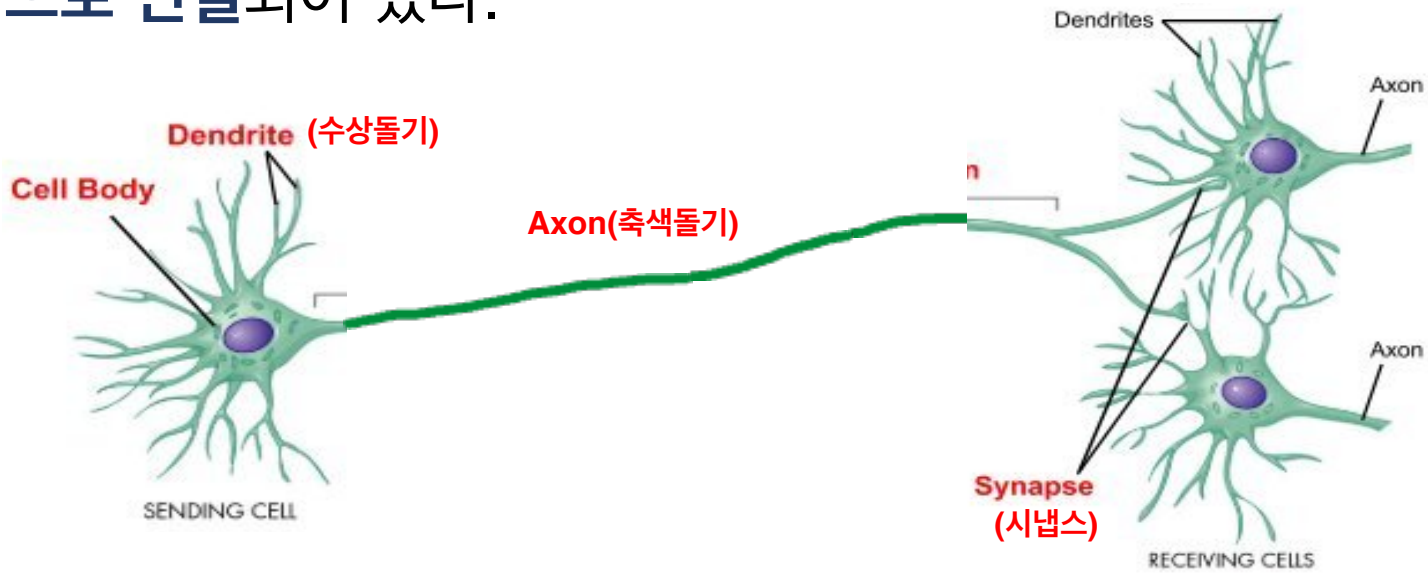
선택 AND (Chice AND)



- ❑ 컴퓨터 공학과를 졸업한 B는 삼성 SDI와, 삼성 SDS 중 **삼성 SDS에 지원**한다.
- ❑ B는 인사담당자인 A에게 삼성 SDS에 지원한다고 말한다.

생물학적 뉴런

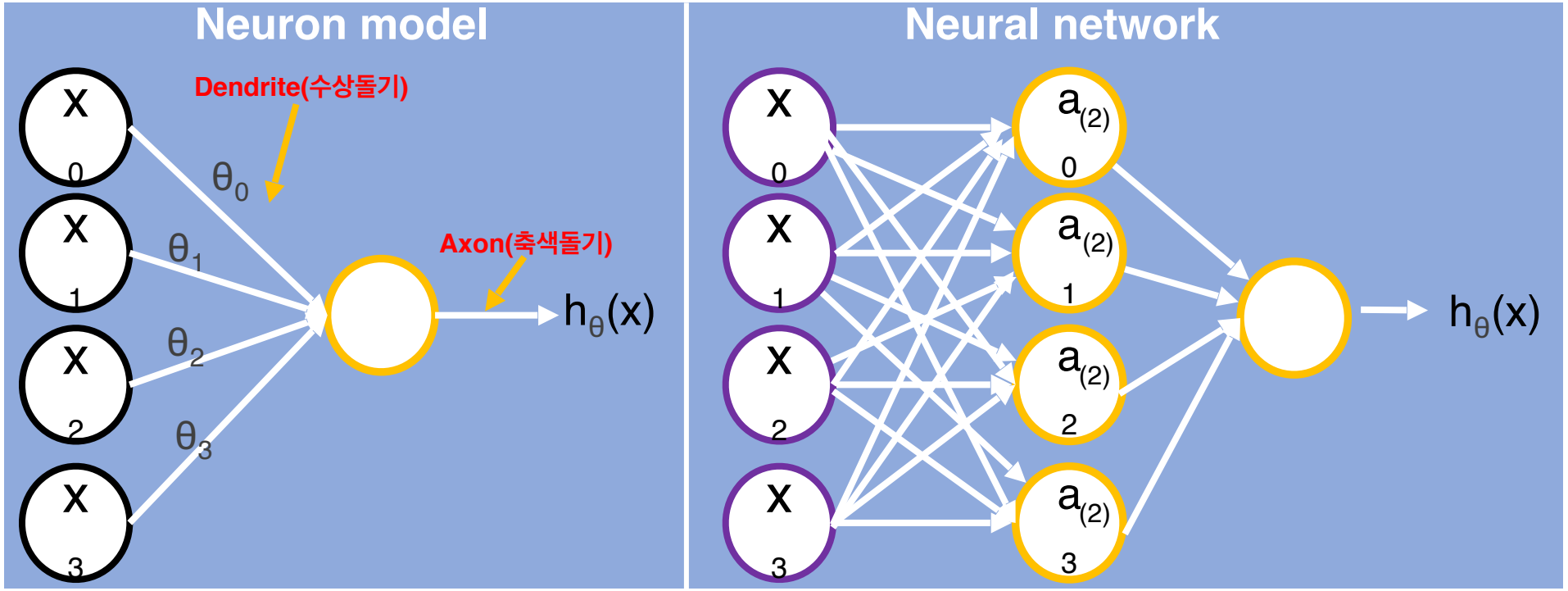
- **인간의 뇌**는 1000억 개가 넘는 신경세포가 100조개의 시냅스를 통해 **병렬적으로 연결**되어 있다.



- 각각의 뉴런은 **수상돌기**를 통해 다른 뉴런에서 **입력신호를 받아**서 **축색돌기**를 통해 다른 뉴런으로 **신호를 보낸**다.
- **시냅스**는 **뉴런과 뉴런을 연결**하는 역할을 한다.

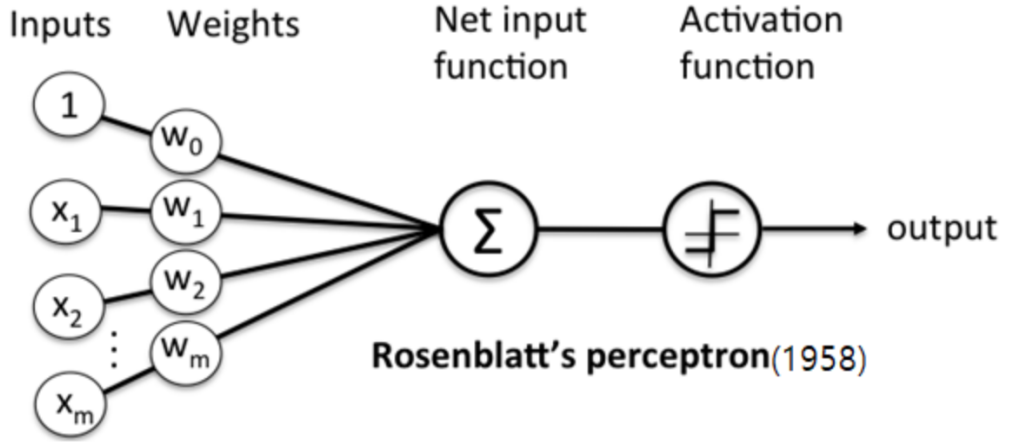
인공신경망(Neural Network)

- 생물학적 **뉴런이 연결된 형태를 모방**해 **수학적**으로 **모델링** 한 것
- **여러 개의 입력** 값을 받아서 **일정 수준이 넘어서**면 **출력** 값을 내보냄
- **인공 신경망**은 뉴런 모델을 **여러 개 쌓아서 만든** 것이다.



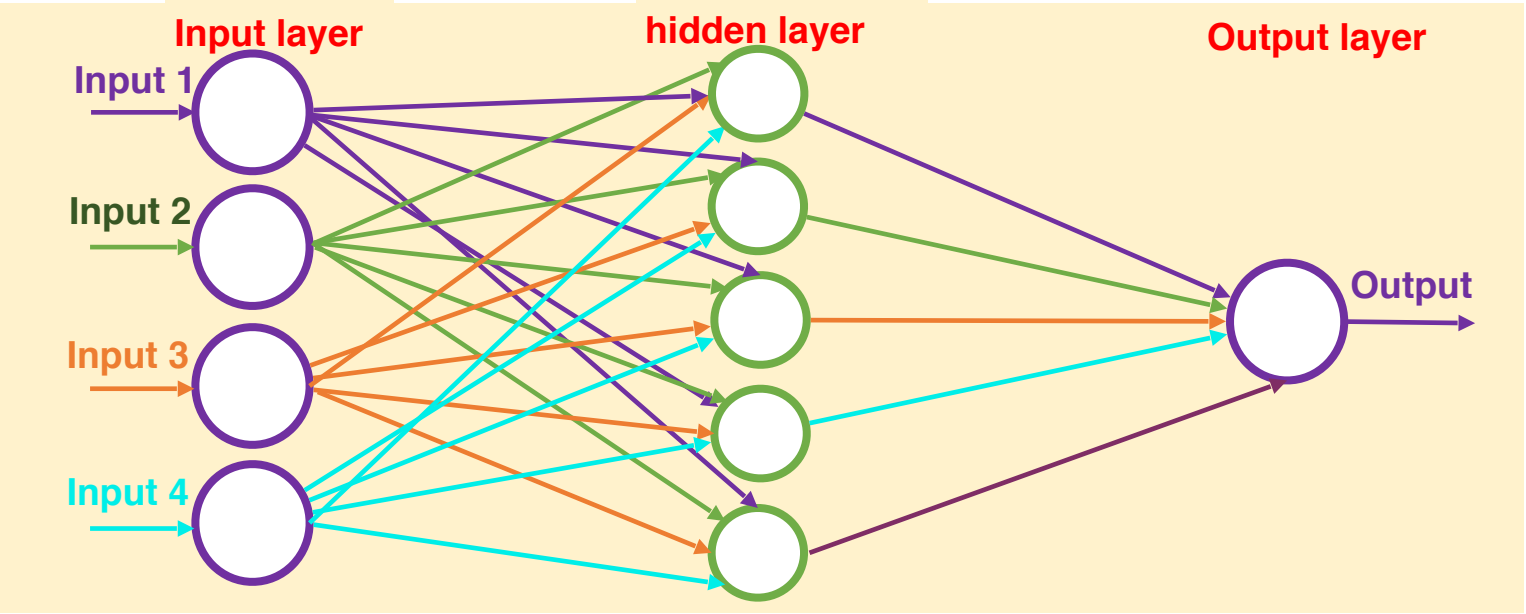
퍼셉트론(Perceptron)

- 1958년 Rosenblatt에 의해 제안된 **선형 분류기**이다.



- **입력과 가중치들의 값**을 모두 더한 후 **활성화 함수**를 적용해 값이 **0 보다 크면 1, 0보다 작으면 -1**을 출력하는 구조다.
- 1969년 마빈 민스키와 Seymour Papert 에 따르면 퍼셉트론은 **단순한 선형 분류기**이며, AND, OR 연산은 할 수 있으나 **XOR 연산을 할 수 없다**.
- 단순한 문제는 풀 수 있으나 **복잡한 문제는 풀 수 없다**는 것이 수학적으로 증명되었다.

MLP(Multi Layer Perceptron)



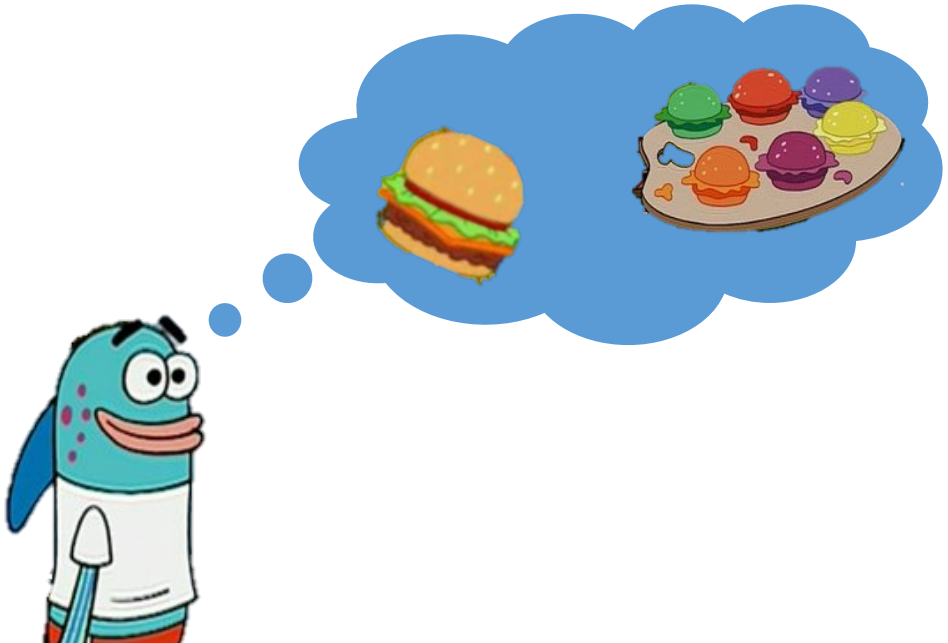
- 1986년에 **은닉계층**이라는 **중간층을 추가**하여 **선형 분류기라는 한계를 극복**한 **다중 퍼셉트론**이 제시되었다.
- 퍼셉트론을 **여러 층으로 쌓으**면서 **학습**하기가 **어려운 문제**가 발생하였는데 이를 **역전파 알고리즘으로 해결**하였다.

** 역전파 알고리즘 = 순방향 연산 후, 예측값과 정답사이의 오차를 후방으로 보내 학습하는 방법

- 즉 **MLP를 통해 XOR 연산**을 할 수 있게 되었다.

NN과 동아 설계도

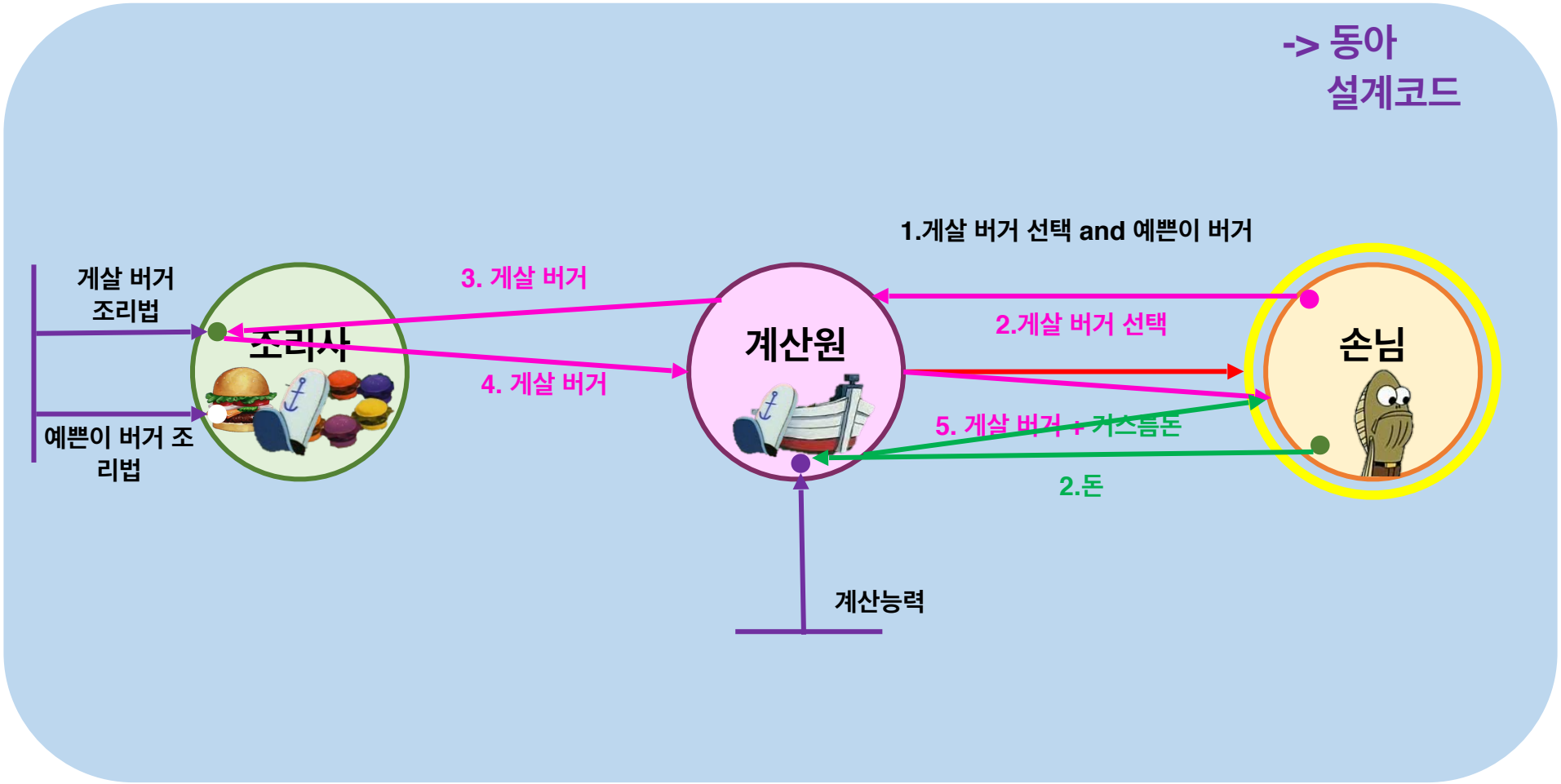
- 주문할 수 있는 메뉴가 2개인 햄버거 가게가 있다. 햄버거 가게를 AI화 시킬 예정이다. 어떻게 설계할 수 있는가?또한 메뉴 하나가 단종 되었을 때는 어떻게 설계할 수 있는가 생각해 보도록 하자.
- **뉴럴 네트워크**는 숫자만 이동한다. 돈의 흐름은 설계할 수 있다.
- **서비스를 구현하기에는 적합하지 않다**.



NN과 동아 설계도

: 객체 : 시작객체 : 서비스 : 능력

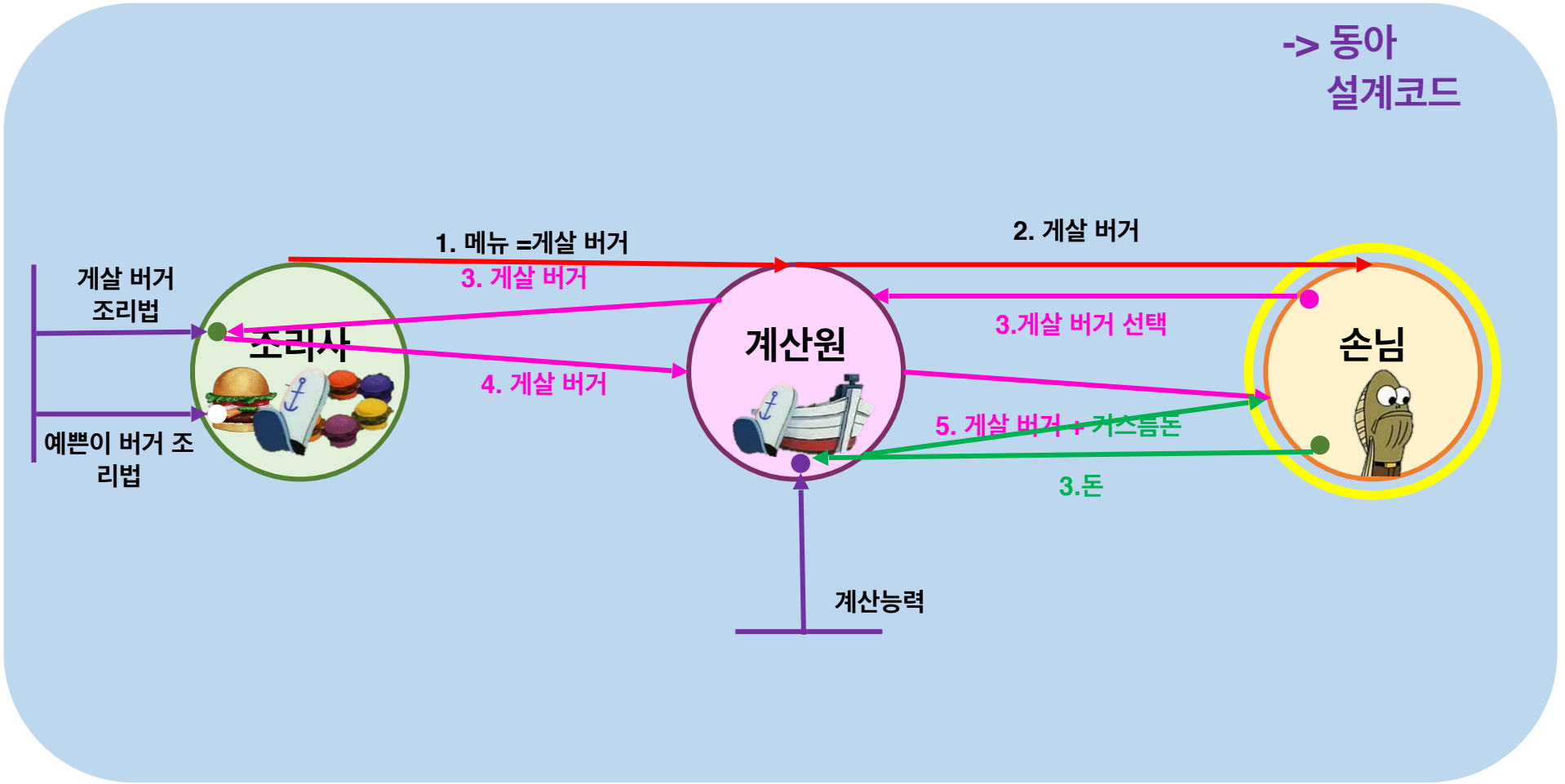
❑ Case1. 메뉴 단종 전 게살 버거 주문 시



NN과 동아 설계도

: 객체 : 시작객체 : 서비스 : 능력

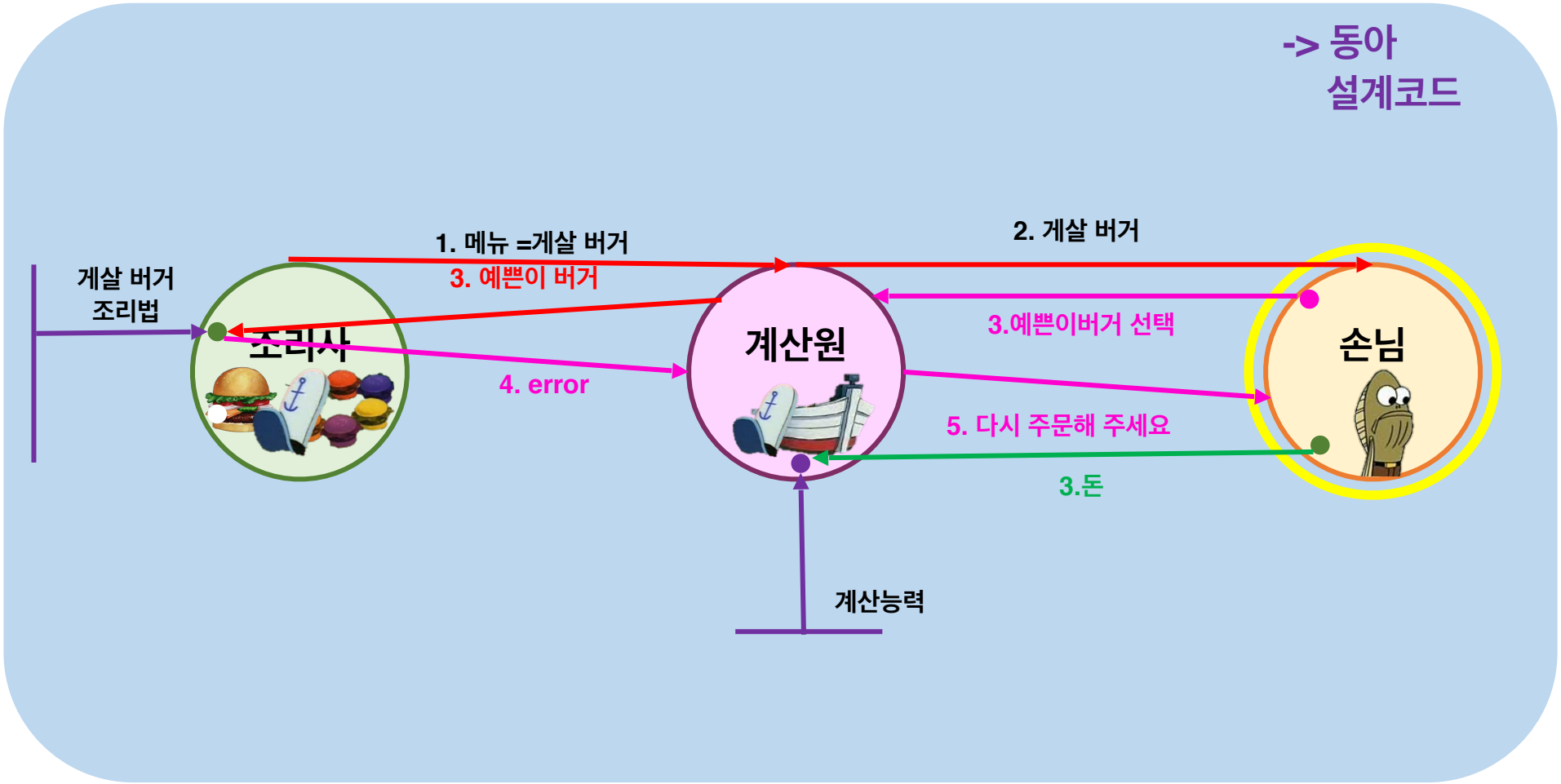
❑ Case2. 메뉴 단종 후 게살 버거 주문 시



NN과 동아 설계도

: 객체 : 시작객체 : 서비스 : 능력

❑ Case3. 메뉴 단종 후 예쁜이 버거 주문 시



NN과 동아 설계도

❑ 공통점

- **여러 층으로 쌓여 있는 형태**이다.
- 그림으로 표현할 수 있다.

❑ 차이점

- NN은 **단순한 데이터(숫자)를 전달**한다.
- 동아 설계코드는 서비스를 제공하고 전달할 수 있다.
- 동아 설계코드는 **복잡한 과정이 필요 없다**.

비자율주행

- 운전시 결정권이 **운전자**에게 있는 경우
  - 행동을 결정하는 주체가 우리들

- **비자율주행 코딩**
  - 기존의 C언어 코딩처럼 **결정권이 프로그래머에게 있는 코드**
  - 프로그래머가 머리를 써야 함
  - 코드가 길어지면 **실수가 발생할 수 있음**
  - 사람이 코드에 영향을 주기 때문에 **안전성이 낮음**

비자율주행

- 비자율주행 코딩의 예시
  - 팩토리얼 함수의 의사코드

```
array fac ;

fac(0) = 1;
for i =1 to n {
        fac(i) = i*fac(i-1);
}
```

- fac(0) =1; 과 같은 것을 할당문이라 한다
- i의 값은 사람에 의해 정해진다
- 정확성과 안정성에 문제가 발생할 수 있다

자율 주행

- 운전시 결정권이 **자동차**에게 있는 경우
  - 행동을 결정하는 주체가 컴퓨터

- **자율주행 코딩**
  - 값을 찾는 방식을 **컴퓨터가 대신하는 코드**
  - 기존 코드에서 프로그래머의 역할을 컴퓨터가 대신함.
  - 잘 짜면 비자율주행 **코드보다 실수가 적음**
  - 컴퓨터가 코드에 영향을 주기 때문에 **안전성이 높음**

자율주행

- 자율주행 코딩의 예시
  - 팩토리얼 함수의 의사코드

```
array fac;

fac(0) = 1;
for i = 1 to n {
        fac(i) = i*fac(i-1);
}

write(x) (x==f(3));
```

- write(x) (x==f(3))을 코드에 추가한다
  - X의 값은 f(3)으로 지정된다
  - I의 값은 컴퓨터에 의해 3으로 지정된다

자율주행과 비자율주행 코드의 차이

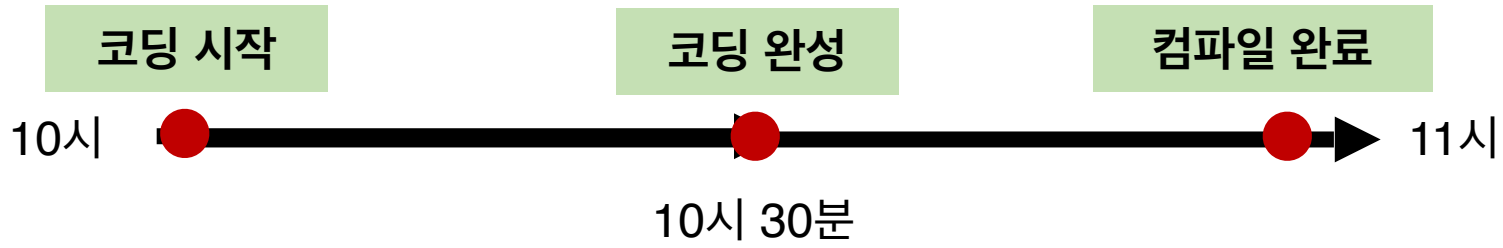
- 값을 결정하는 **주체**의 차이
  - 비자율주행 코드는 행동을 결정하는 주체가 **사람(프로그래머)**
  - 자율주행 코드는 행동을 결정하는 주체가 **컴퓨터**

- **소요 시간**의 차이
  - 자율주행 코드에는 **근거**가 존재하기 때문에 **다른 사람이 보고 분석 가능**
  - 협업에서 근거가 있는 방식으로 코드를 짜면 더 빠르게 코드가 완성되기 때문에 **자율주행 코드가 비자율주행 코드보다 빠르게 완성**됨

두 코드의 소요시간 차이 예시

- 비자율주행 코드
  - 10시에 코드를 짜기 시작하면 11시에 코딩이 완료되고 컴파일을 한다

- 자율주행 코드
  - 10시에 코드를 짜기 시작하면 10시 30분에 코딩이 완료되고 11시에 컴파일까지 끝낼 수 있다



동아 설계도를 구현하기에 적합한 코드

- 설계도에 따라 **수백만 줄의 코드가 필요**
  - 설계가 완벽하더라도 **코드의 안전성과 정확성이 떨어지면 문제가 생김**
  - 비자율주행 코드는 줄이 긴 코드에서 불리함

- 자율주행 코드가 **협업에서 더 유리**
  - 근거가 있는 코딩 방식을 사용하기 때문에 다른 사람이 설계도를 구현한 내용을 더 쉽게 알아볼 수 있음
  - 혼선이 발생할 일이 줄어 협업이 순조롭게 이루어짐

따라서, 자율주행 코드로 동아 설계도를 구현해야 한다

UML(Unified Modeling Language)

- **1997년 OMG(Object Management Group)에서 표준으로 채택한 통합 모델링 언어**
- 가장 **보편적으로 쓰이는 표준이며 객체 지향 시스템**을 가시화, 명세화, 문서화하기 위해 사용되며 **사물, 관계, 다이어그램으로 구성**
- 시스템의 여러 측면을 **그림으로 모델링**
- **하드웨어의 회로도** 같은 **의미**를 가짐

- UML의 장점
  - **복잡한 구조**를 **단순화**하고 **원활한 의사소통**을 도움
  - 업무의 질이 향상하고 비용 절감 및 시장 출시 시간 단축 효과를 가짐

기존 UML의 규칙

- 가장 윗부분: 클래스의 이름
- 중간부분: 속성(클래스의 특징)
- 마지막 부분: 연산(클래스가 수행하는 책임)
- 접근 제어자, 표시, 설명은 밑의 그림을 확인해보자

접근 제어자	표시	설명
public	+	어떤 클래스의 객체에서든 접근 가능
private	-	이 클래스에서 생성된 객체들만 접근가능
protected	#	이 클래스와 동일 패키지에 있거나 상속 관계에 있는 하위 클래스의 객체들만 접근 가능
package	~	동일 패키지에 있는 클래스의 객체들만 접근 가능

UML의 구성 요소 - 사물

- **사물(Things)**
  - 다이어그램 안에서 **관계가 형성될 수 있는 대상**
  - 구조 사물, 행동사물, 그룹사물, 주해사물로 나뉨

명칭	설명	종류
구조 사물	시스템의 개념적, 물리적 요소를 표현	클래스, use case, 컴포넌트
행동 사물	시간과 공간에 따른 요소들의 행위를 표현	상호작용, 상태 머신 등
그룹 사물	요소들을 그룹으로 묶어서 표현	패키지
주해 사물	부가적인 설명이나 제약조건 등을 표현	노트

UML의 구성요소 - 관계

❑ 관계(Relationships)

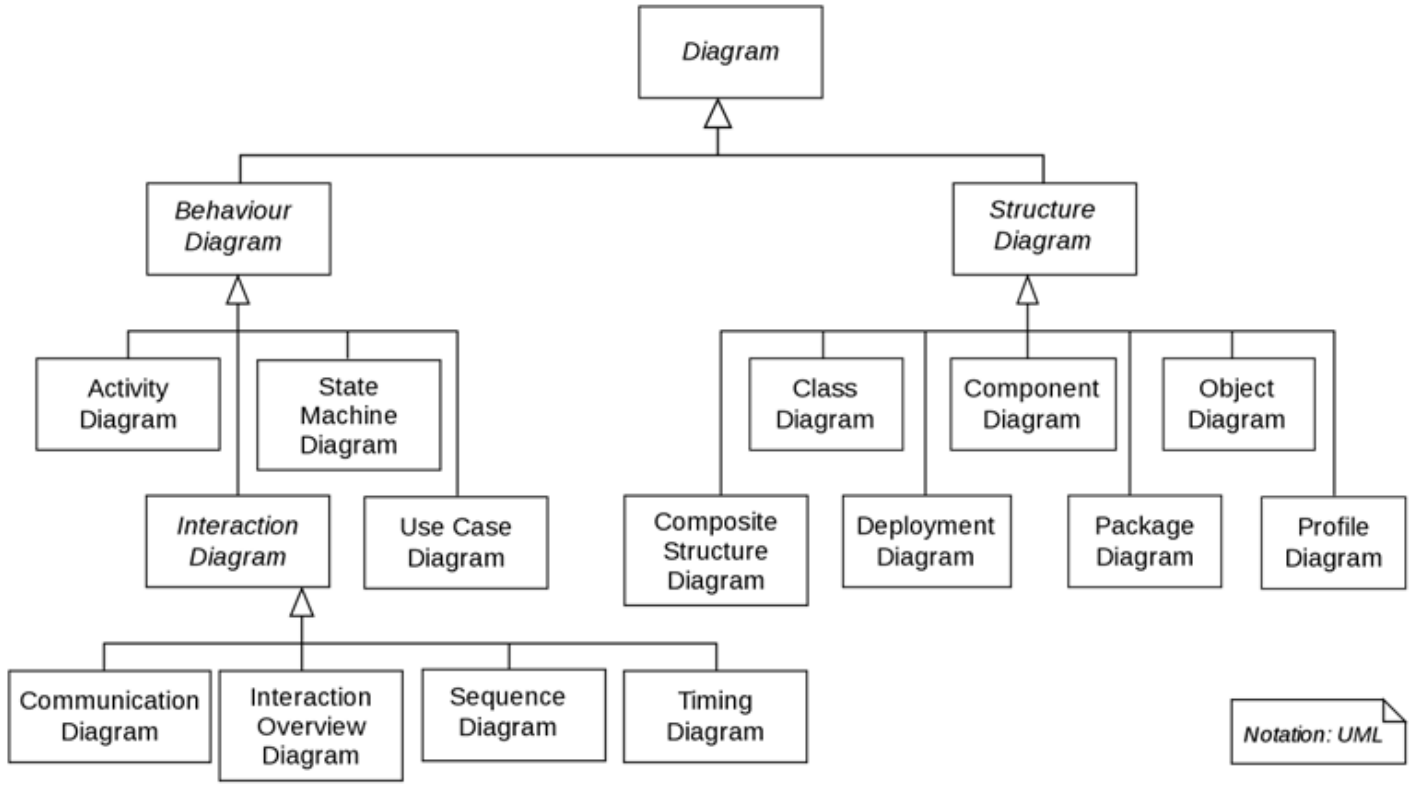
- ❑ **사물과 사물 사이의 연관성**
- ❑ 연관관계,집합관계,포함관계,일반화 관계,의존관계,실체화관계

명칭	표현	설명
연관 관계		객체간 개념적으로 연관
의존 관계		객체 변경 시 관계된 다른 객체도 변경
집합 관계		전체와 부분인 연관 관계
포함 관계		전체 소멸 시 부분도 소멸되는 집합 연관 관계
일반화 관계		일반화된 개념과 구체화된 객체의 관계

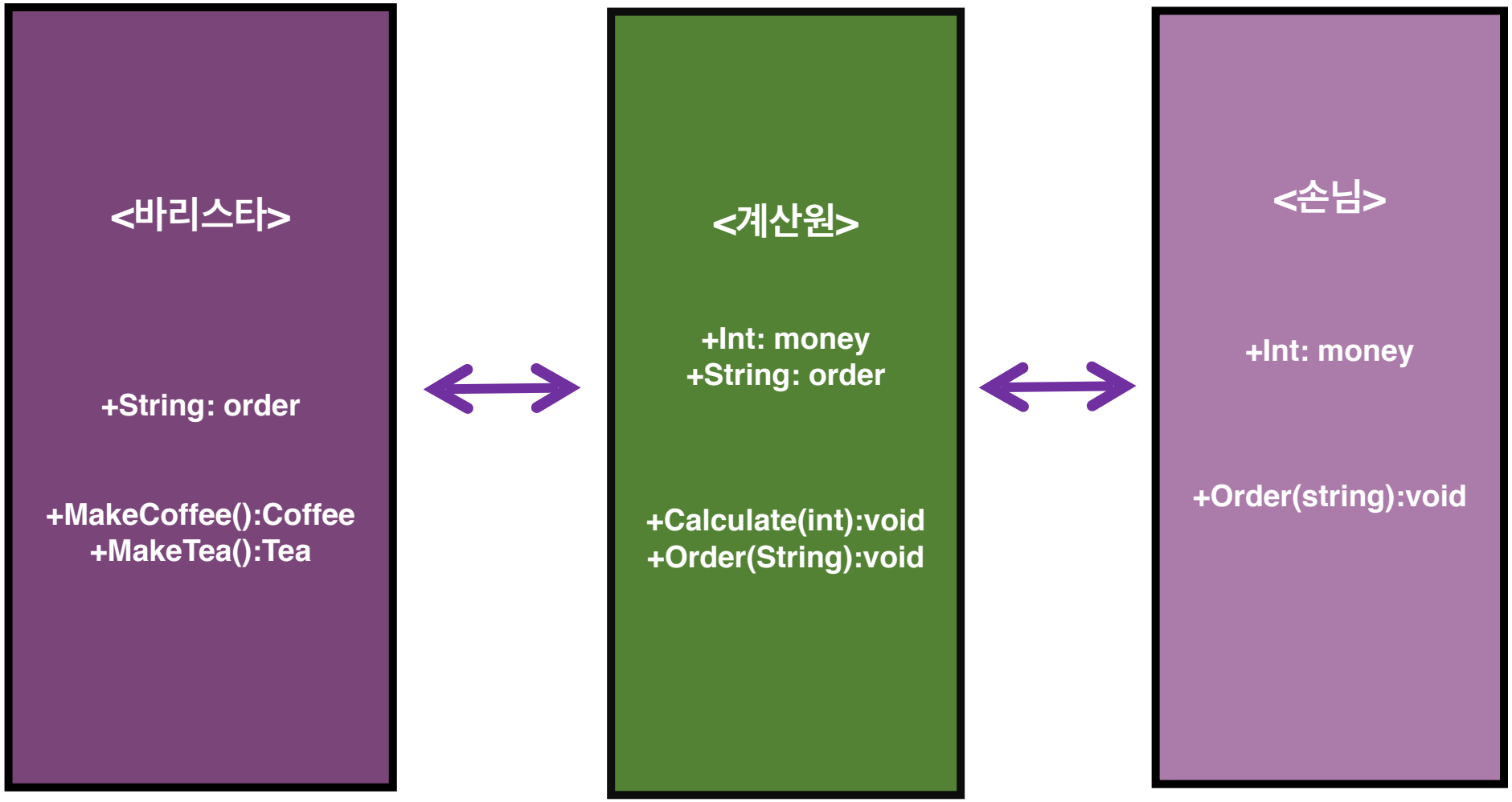
UML의 구성 요소 - 다이어그램

❑ 다이어그램(Diagram)

- ❑ 시스템 **상호작용 및 구조, 업무 흐름, 컴포넌트 간의 관계 등을 그린 도면**
- ❑ 크게 구조적 다이어그램과 행위 다이어그램으로 나뉨

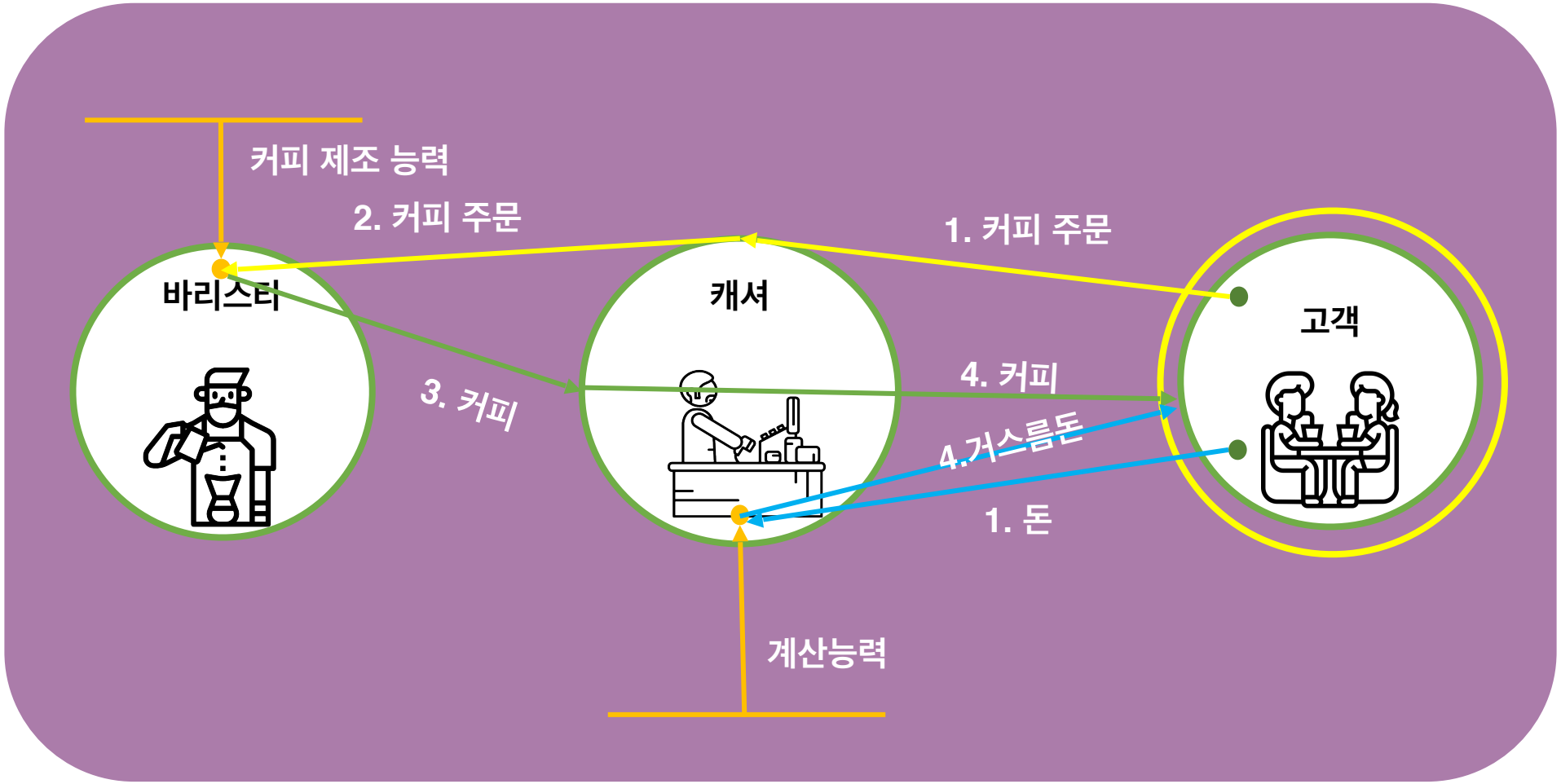


UML로 표현한 커피숍.



동아 설계코드 UML.

: 객체 : 시작 객체 : 서비스 : 능력



동아 설계도 vs IBM UML

- 동아 설계도는 한국어를 사용할 수 있음
- 동아 설계도는 우리가 만들었기에 무료로 사용 가능
- IBM사의 **UML 만큼 편리하게 사용** 가능함
- 동아 설계도는 **요소들이 내부 정보들의 관계를 표현하도록 설계** 가능
- 동아 설계도는 **선택 OR, 선택 AND 등 더 다양한 관계를 표현할 수 있음**

참고문헌

❑ 참고문헌

- Axon Dendrite Cell Body Neuron (Nerve cell) Anatomy Synapse
- Biological neuron model - Wikipedia
- Artificial neural network - Wikipedia
- https://github.com/kkyoulza - GitHub
- Keehang Kwon Hyungjoon Kwon, 2022, "Logical Pseudocode: Connecting Algorithms with Proofs", 1~3.
- 솔라리스의 인공지능 연구실 - 딥러닝의 역사
- 7. 소프트웨어공학(1~14주차) - 권기항 교수님2022년 소프트웨어 공학 (권기항) 강의