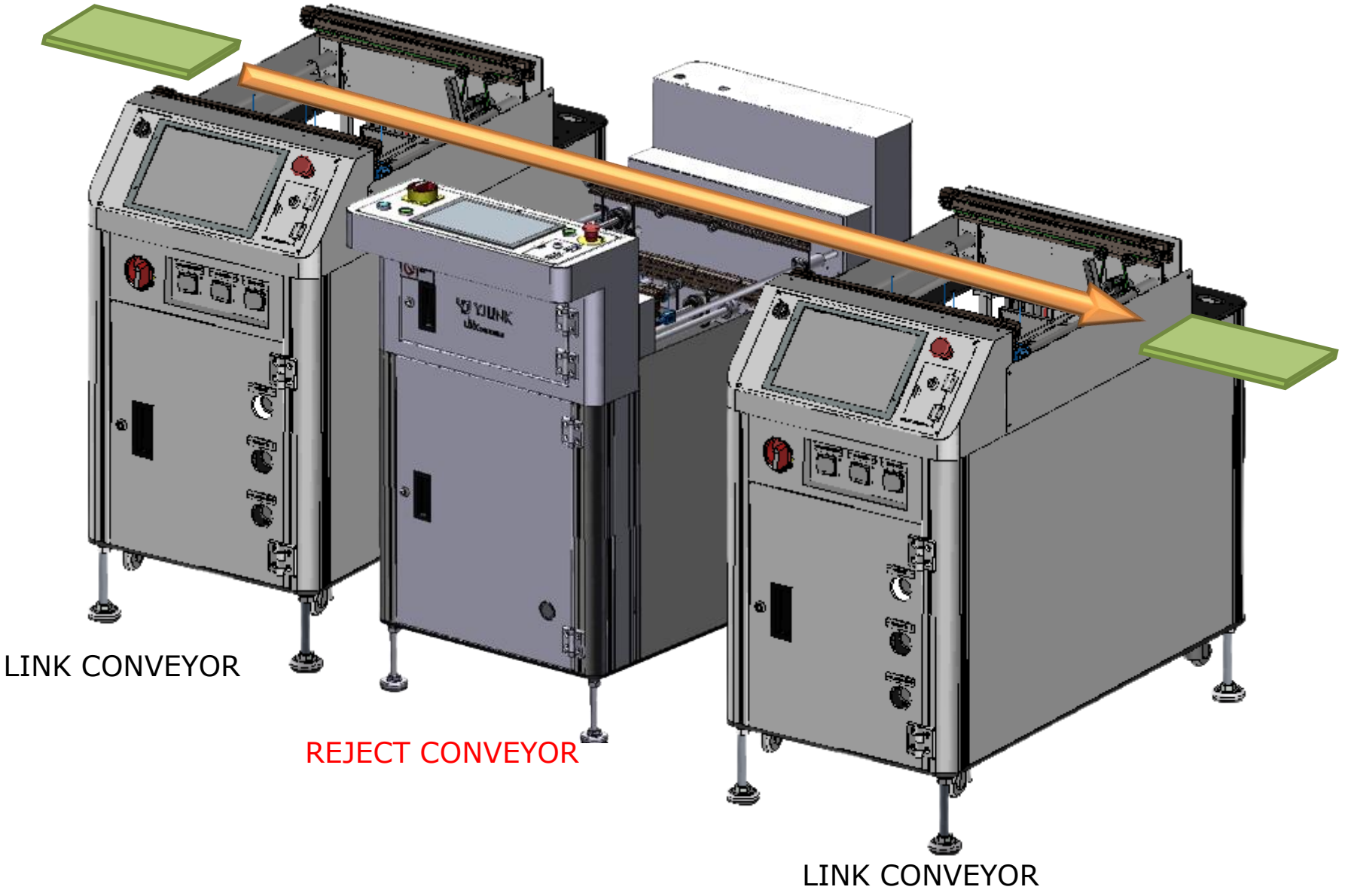
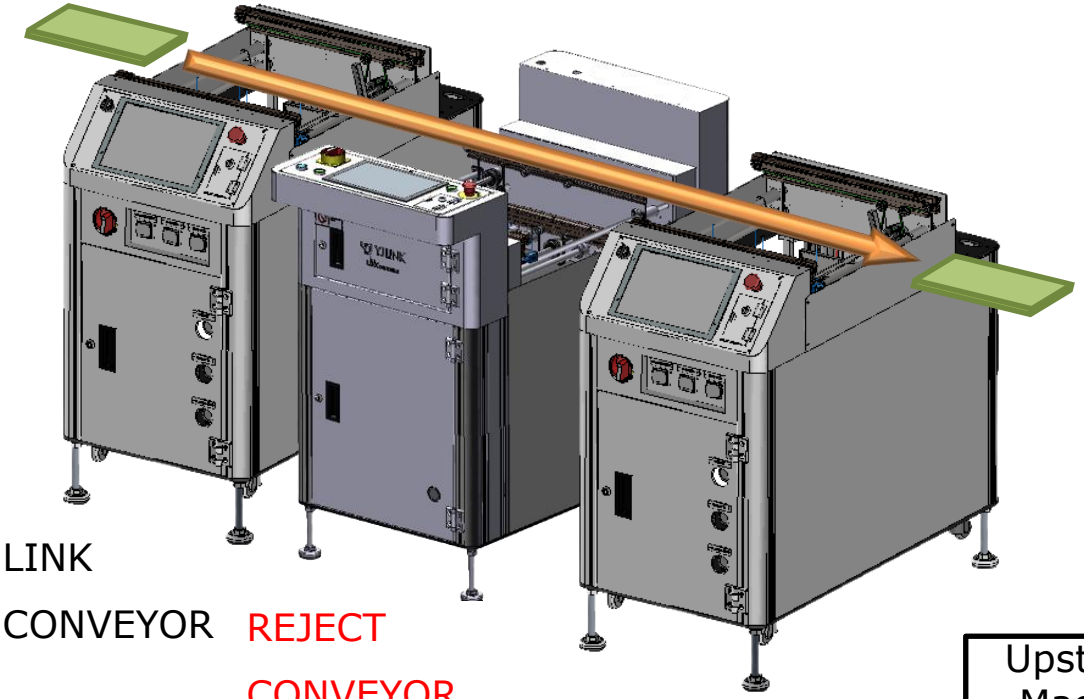
시스템 설명



SMEMA Interface : IPC-SMEMA-9851



LINK CONVEYOR

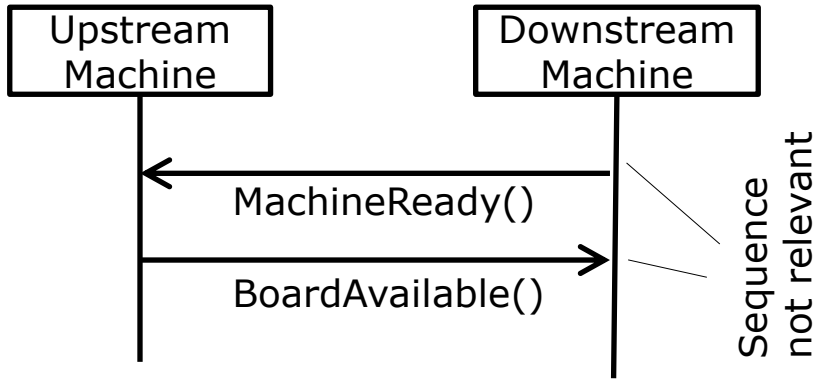
REJECT CONVEYOR

LINK CONVEYOR

SMEMA Connector

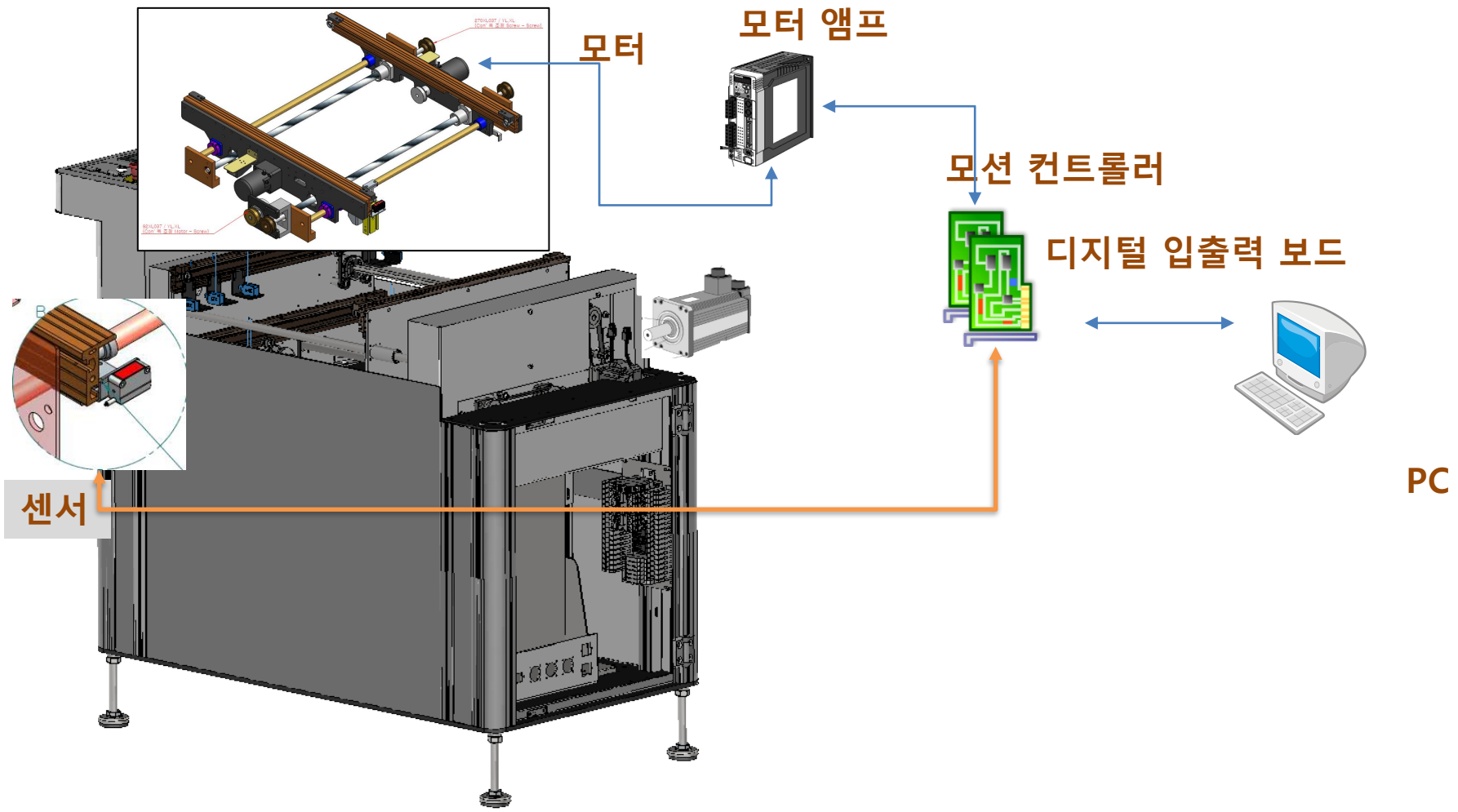
전단 설비 연결

후단 설비 연결



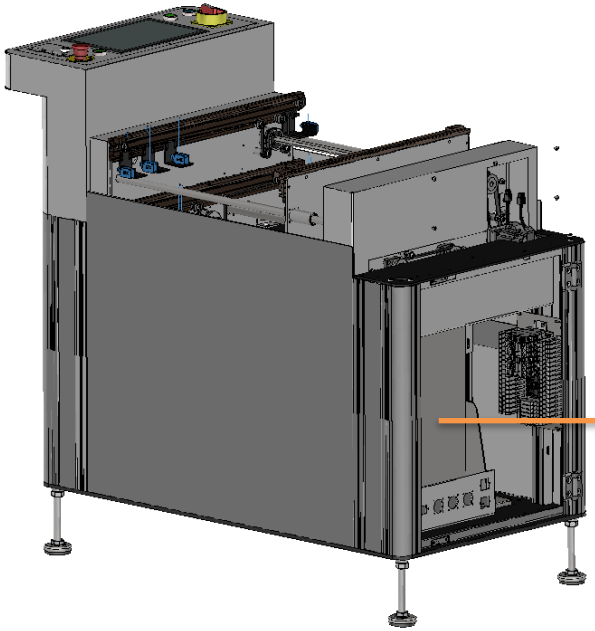
후단의 Machin Ready 신호를 받은 후
전단 장비에서 보드 배출

제어 부품의 구성

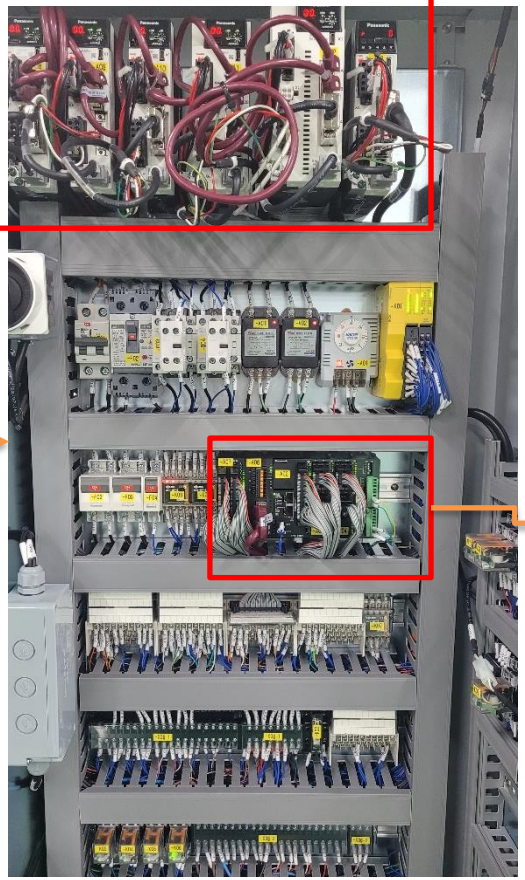


제어 프로그램의 작성 = 모터와 입출력 신호를 이용하여 설비의 제어 시퀀스 작성

설비 제어 시스템



모터 앰프



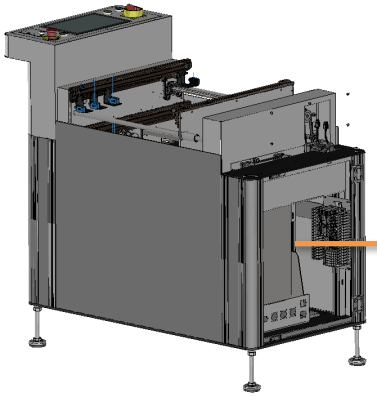
모션 컨트롤러

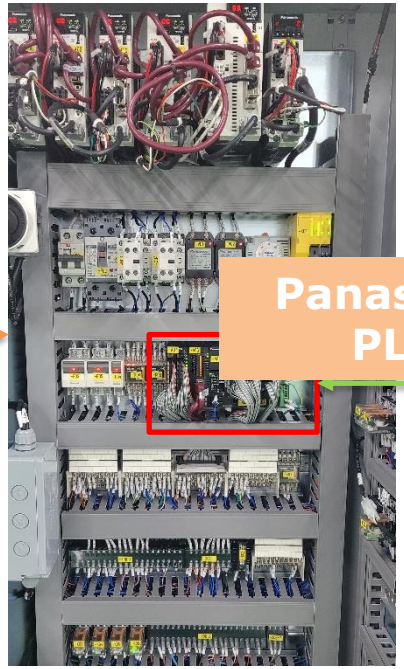
디지털 입출력 보드

소프트웨어 실습 순서

1	Motion Board/IO Board 제어	Mewtocol 통신 작성
2	입출력 신호 제어	IO Read/Write
3	Windows GUI 작성	
4	Reject conveyor 제어	
5	객체 지향 S/W 의 작성	
6	다형성	
7	State Pattern	
8	Strategy Pattern	
9	Hermes 작성	Machine to Machine I/F

Motion Board/IO Board 제어 – UDP 통신 모듈 작성





Mewtocol

PC

통신 Type - UDP

Protocol - Mewtocol

Mewtocol 작성 – PLC Device 제어

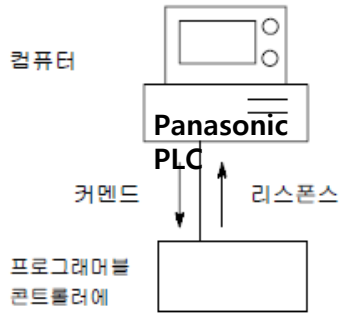
PLC IP Address: 192.168.1.5, Port number: 8000

PLC IP Address: 127.0.0.1, Port number: 8000

1.1.1 MEWTOCOL－COM의 개요

■ 커 멘 드 / 리 스 폰 스 의 기 능

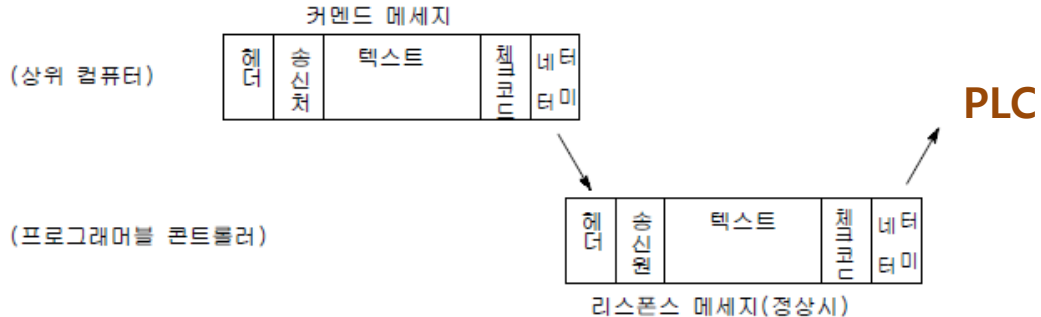
컴퓨터는 프로그래머블 콘트롤러에 대해서 커멘드(명령)를 보내, 리스폰스(대답)를 받습니다. 이 순서에 의해 컴퓨터는 프로그래머블 콘트롤러에 대해서 회화가 행 네, 각종 정보를 얻거나 주거나 할 수가 있습니다.



주의 :
컴퓨터 링크를 하기에는 컴퓨터측의 유저프로그램만 필요합니다.
프로그래머블 콘트롤러측의 프로그램은 필요로 하지 않습니다.

PC

■ 커 멘 드 / 리 스 폰 스 의 포 맷



Mewtocol (Protocol) 설명

● 제어 코드

명칭	캐릭터	ASCII코드	설명
헤더	% 또는 <	25H또는 3CH	메세지의 개시를 나타낸다.
커멘드	#	23H	커멘드 · 메세지인 것을 나타낸다.
리스폰스(정상)	$	24H	정상적인 리스폰스 · 메세지인 것을 나타낸다.
리스폰스(이상)	!	21H	에러시의 리스폰스 · 메세지인 것을 나타낸다.
터미네이터	C_R	0DH	메세지의 종료를 나타낸다.
딜리미터	&(+C_R)	26H	복수 프레임에 분할할 때의 구별을 나타낸다.

커멘드 메세지

헤더	송신처	텍스트	체크코드	터미네이터

● 송신처,송신원 **AD (H),(L)**

2자리수의 10진수 01~32(ASCII코드)
커멘드 메세지내에서는, 커멘드 메세지를 수취해야 할 프로그래머블 콘트롤러의 UNIT No. 를 나타냅니다.
리스폰스 메세지내에서는, 리스폰스 메세지를 송출한 PC의 UNIT No. 를 나타냅니다.
(H)는 위의 자리수, (L)는 아래의 자리수를 나타냅니다.
특히 지정이 없으면, "01"과 지정해 주세요.
다만, FF(ASCII 코드값)시에는, 글로벌 전송(전유니트에의 일제 전송※)입니다.
주※ 글로벌 전송을 할 경우, 그 커멘드 메세지에 대한 리스폰스 메세지는 반송하지 않습니다.

● 블록 체크 코드 **BCC (H),(L)**

2자리수의 16진수 00~FF(ASCII코드)
전송 데이터의 잘못 검출용의 코드(수평 패리티)입니다.
다만, BCC 대신에**를 넣을 경우는, BCC 없음으로 전송이 가능합니다.
이 경우도 리스폰스에는 BCC가 붙어 옵니다.

Mewtocol 작성 – 단일 접점 읽기

1. 단일 접점을 읽는 명령어 – X00A 접점 값을 읽는 경우

예) <01#RCSX000A**Cr

ASCII 값 0x0D

터미네이터	C_R	0DH	메세지의 종료를 나타낸다.

[RCS] 접점 에리어 리드(단점)

접점의 ON/OFF 상태를 일점만 읽어냅니다.

■커멘드

% 또는 <	송신처 ×10¹ ×10⁰	#	R	C	S	접점 코드 1문자	접점No. 4문자 ×10³ ×10² ×10¹ ×10⁰ (×10⁰)	BCC ×16¹ ×16⁰	C_R

S: 단점취급

접점No.: ( ) 안은, TM/CT의 경우입니다.

■정상시 리스폰스(리드 OK)

% 또는 <	송신원 ×10¹ ×10⁰	$	R	C	접점 데이터 1문자	BCC ×16¹ ×16⁰	C_R

■에러 리스폰스(read error)

% 또는 <	송신원 ×10¹ ×10⁰	!	에러 코드 ×16¹ ×16⁰	BCC ×16¹ ×16⁰	C_R

● 에러 코드 **Err (H),(L)**

2자리수의 16진수 00~FF(ASCII코드)
에러 발생시에 그 내용을 나타냅니다.

접점 코드

접점		표기
외부 입력	X	"X"
외부 출력	Y	"Y"
내부 릴레이	R	"R"
링크 릴레이	L	"L"
타이머	T	"T"
카운터	C	"C"

접점 데이터

접점	표기
ON	"1"
OFF	"0"

ASCII Code

제어 문자		공백 문자		구두점		숫자		알파벳			
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	HT	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n

Mewtocol 작성 – 복수접점 읽기

1. 복수 접점을 읽는 명령어
2. Address 에 18을 설정하면 X180 ~ X18F 까지 16bit 값 반환
3. X180 ~ X19F 까지의 값을 읽는 명령어

예) <01#RCCX00180019**Cr -> <01$RCa7b386fe**Cr

X180 의 값은?

주소	Data(16진수)	Data(2진수)
X180 ~ X18F	0xa7b3	1010_0111_1011_0011
X190 ~ X19F	0x86fe	1000_0110_1111_1110

[RCC] 접점 에리어 리드(워드 단위 블록)

접점의 ON/OFF 상태를 워드 단위로 읽어냅니다.

■커멘드

% 또는 <	송신처 ×10¹ ×10⁰	#	R	C	C	접점 코드 1문자	선두 워드 No. 4문자 ×10³ ×10² ×10¹ ×10⁰	최종 워드 No. 4문자 ×10³ ×10² ×10¹ ×10⁰	BCC ×16¹ ×16⁰	C_R

워드 취급

■정상시 리스폰스(리드 OK)　　접점 정보는, 워드 단위에 16 진수에서 읽어내집니다.

% 또는 <	송신원 ×10¹ ×10⁰	$	R	C	접점 정보 (선두) 4문자 ×16¹ ×16⁰ ×16³ ×16²	접점 정보 (최종) 4문자 ×16¹ ×16⁰ ×16³ ×16²	BCC ×16¹ ×16⁰	C_R

(하위) (상위) (하위) (상위)

Little Endian – Big Endian

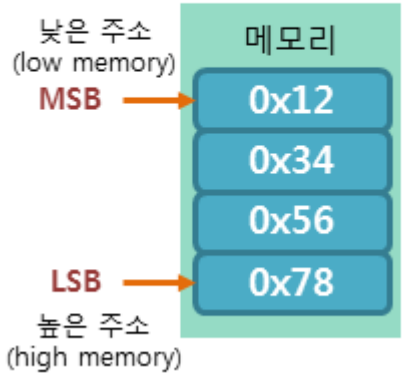
Big Endian - 사람이 숫자를 쓰는 방법과 같이 큰 단위의 바이트가 앞에 오는 방법,

X86 (Intel)

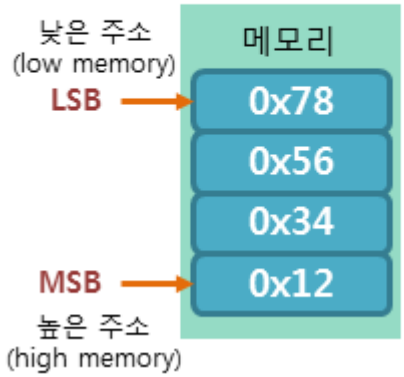
Little Endian - 작은 단위의 바이트가 앞에 오는 방법, Motorola

종류	0x1234의 표현
빅 엔디언	12 34
리틀 엔디언	34 12

- 빅 엔디안(Big Endian)



리틀 엔디안(Little Endian)



Mewtocol 작성 – 단일 접점 쓰기

1. 단일 접점을 쓰는 명령어 – X00A 접점에 1을 쓰는 경우

예) <01#WCSX000A1**Cr

[WCS] 접점 에리어 라이트(단점)

접점을 일점만 ON/OFF 합니다.

■커멘드

% 또는 <	송신처 ×10¹ ×10⁰	#	W	C	S	접점 코드 1문자	접점No. 4문자 ×10³ ×10² ×10¹ ×10⁰ (×16⁰)	접점 데이터 1문자	BCC ×16¹ ×16⁰	CR

단점취급

■정상시 리스폰스(라이트 OK)

% 또는 <	송신원 ×10¹ ×10⁰	$	W	C	BCC ×16¹ ×16⁰	CR

■에러 리스폰스(write error)

% 또는 <	송신원 ×10¹ ×10⁰	!	에러 코드 ×16¹ ×16⁰	BCC ×16¹ ×16⁰	CR

접점 코드

접점	표기
외부 입력 X	"X"
내부 릴레이 R	"R"
링크 릴레이 L	"L"

접점 데이터

접점	표기
ON	"1"
OFF	"0"

Mewtocol 작성 – 데이터 영역 읽기

1. 데이터 레지스터 영역의 데이터를 읽는 명령어 – 10000 ~ 10001 값을 읽는 경우

예) <01#RDD1000010001**Cr -> <01$RD3f1a3d02**Cr

주소	Data
10000	1a3f
10010	23d

[RD] 데이터 에리어 리드

데이터 에리어의 내용을 읽어냅니다.

DT, LD, FL의 내용을 읽어내는 경우

■커멘드

% 또는 <	송신처 ×10¹ ×10⁰	#	R	D	데이터 코드 1문자	선두 워드 No. 5문자 ×10⁴ ×10³ ×10² ×10¹ ×10⁰	최종 워드 No. 5문자 ×10⁴ ×10³ ×10² ×10¹ ×10⁰	BCC ×16¹ ×16⁰	CR

■정상시 리스폰스(리드 OK)

% 또는 <	송신원 ×10¹ ×10⁰	$	R	D	레지스터 내용(선두) 4문자 ×16¹ ×16⁰ ×16³ ×16²	⋯	레지스터 내용(최종) 4문자 ×16¹ ×16⁰ ×16³ ×16²	BCC ×16¹ ×16⁰	CR
					(하위) (상위)		(하위) (상위)		

데이터 코드

데이터	표기
데이터 레지스터 DT	"D"
링크 레지스터 LD	"L"
파일 레지스터 FL	"F"

Mewtocol 작성 – 데이터 영역 쓰기

1. 데이터 레지스터 영역의 데이터를 쓰는 명령어 – **10000 ~ 10001** 값에 값을 쓰는 경우

예) **<01#WDD10000100013f1a3d02**Cr**

주소	Data
10000	1a3f
10010	23d

[WD] 데이터 에리어 라이트

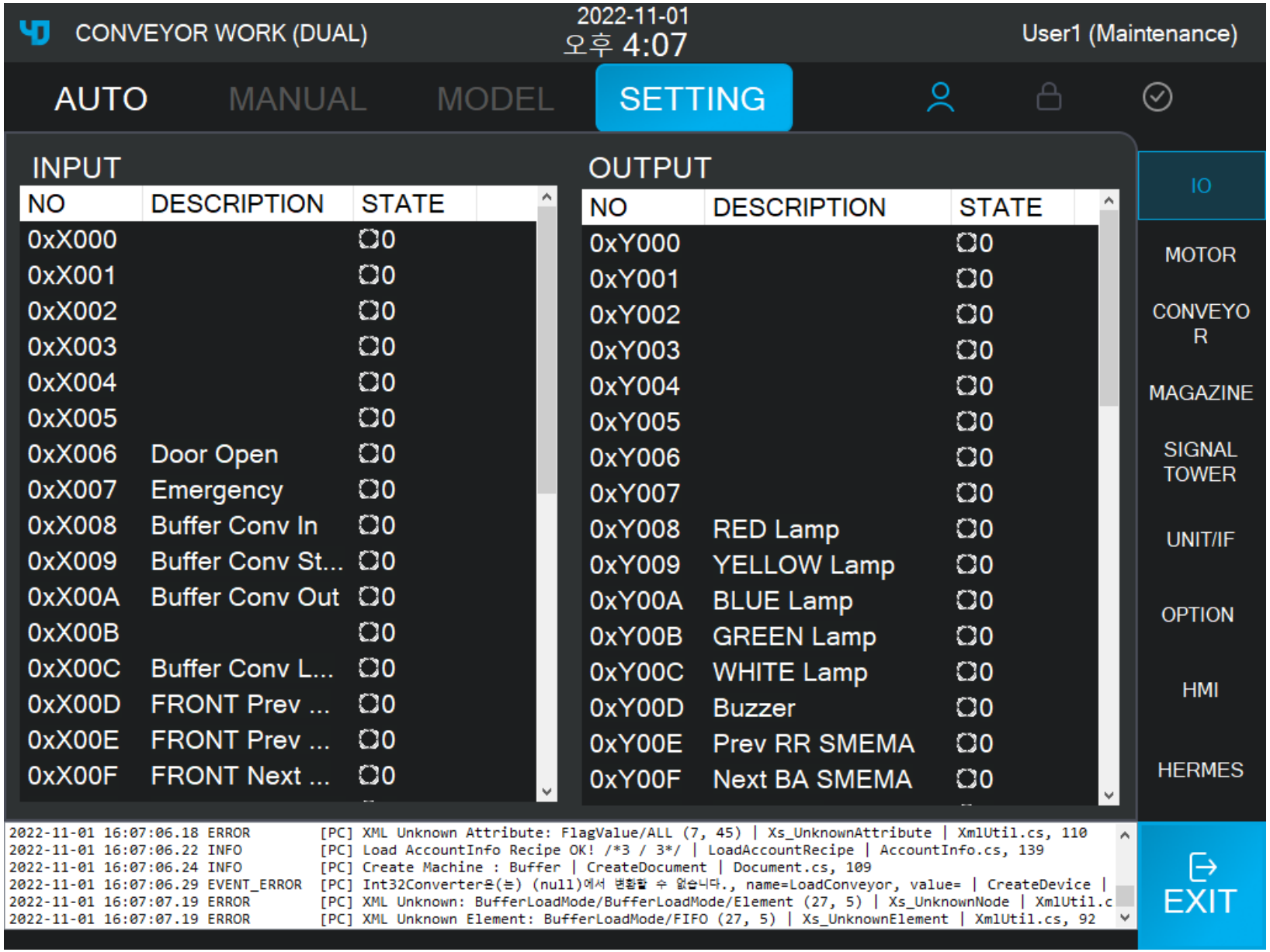
데이터 에리어의 내용을 기입합니다.

DT, LD, FL의 내용을 기입하는 경우

■커멘드

WinForm 으로 프로그램 작성하는 방법

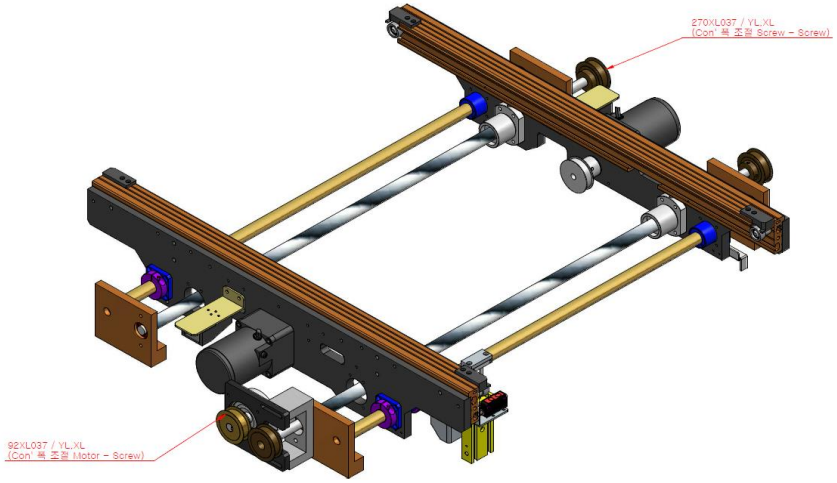
Gui 작성 – IO 를 이용한 수동 동작 제어

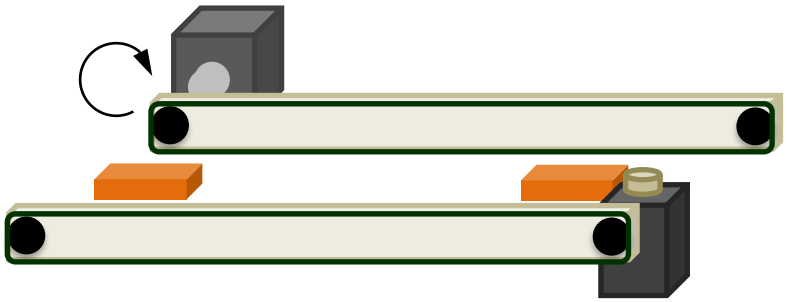


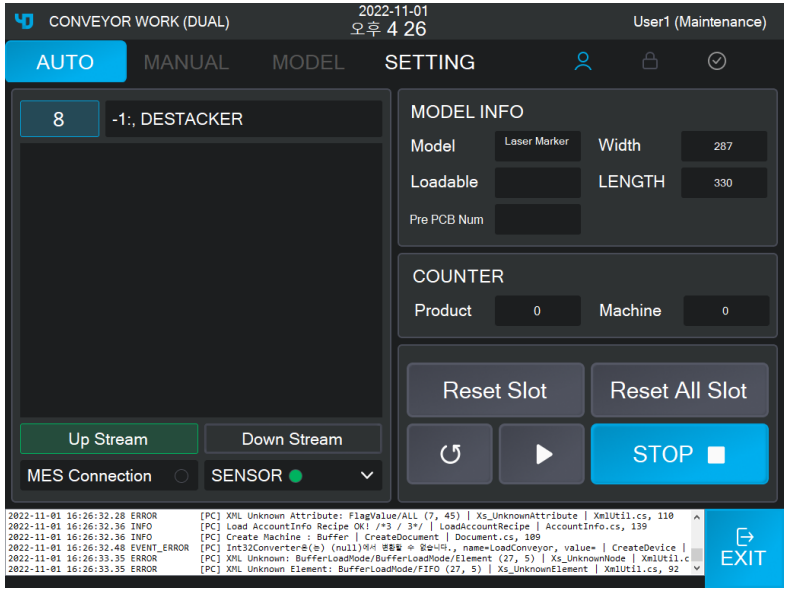
실습 1

1. Winform 프로그램으로 입출력 제어 하기
    1) UDP 통신으로 Mewtocol 구현

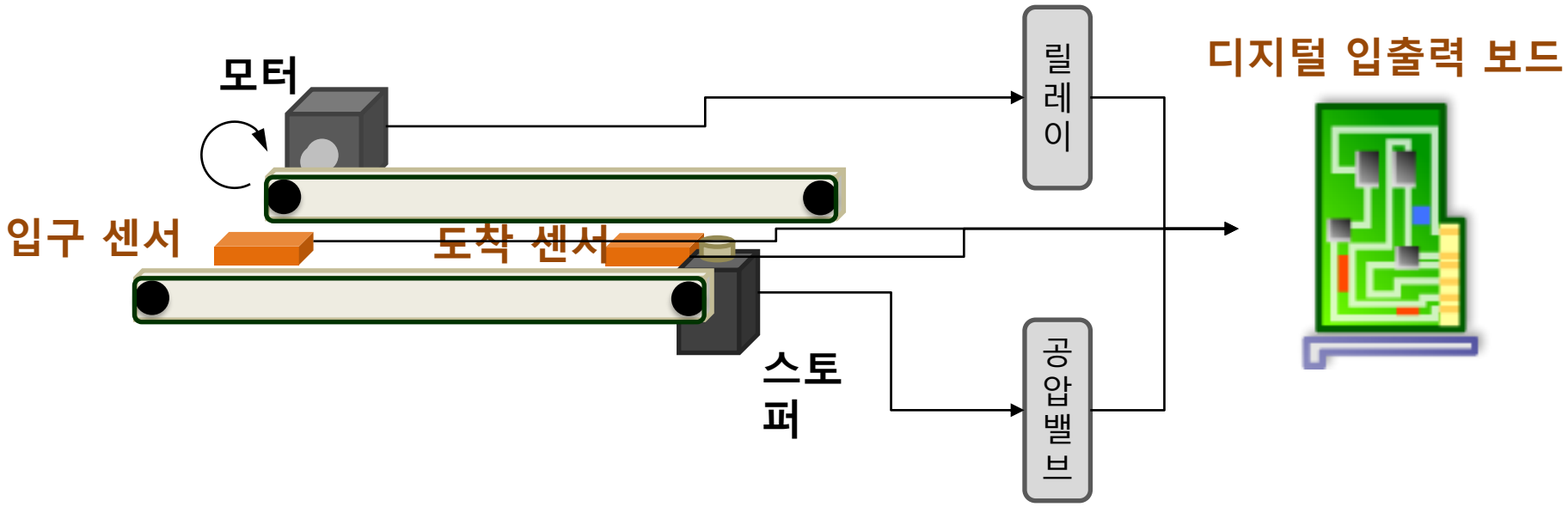
IO 를 이용한 수동 동작 제어

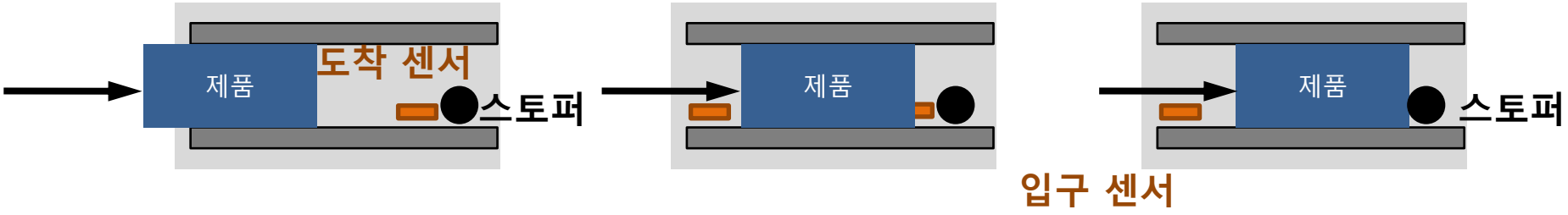






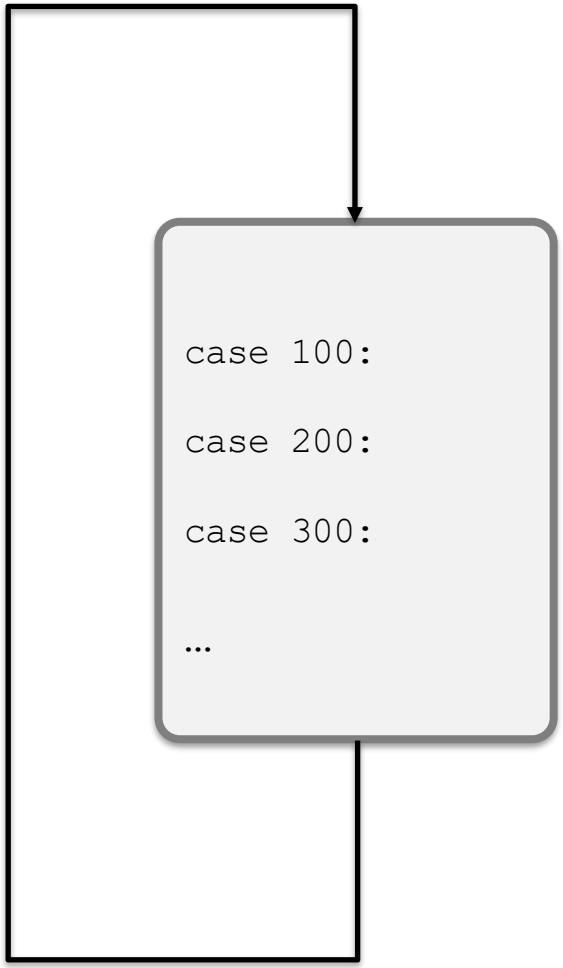
컨베이어 제어

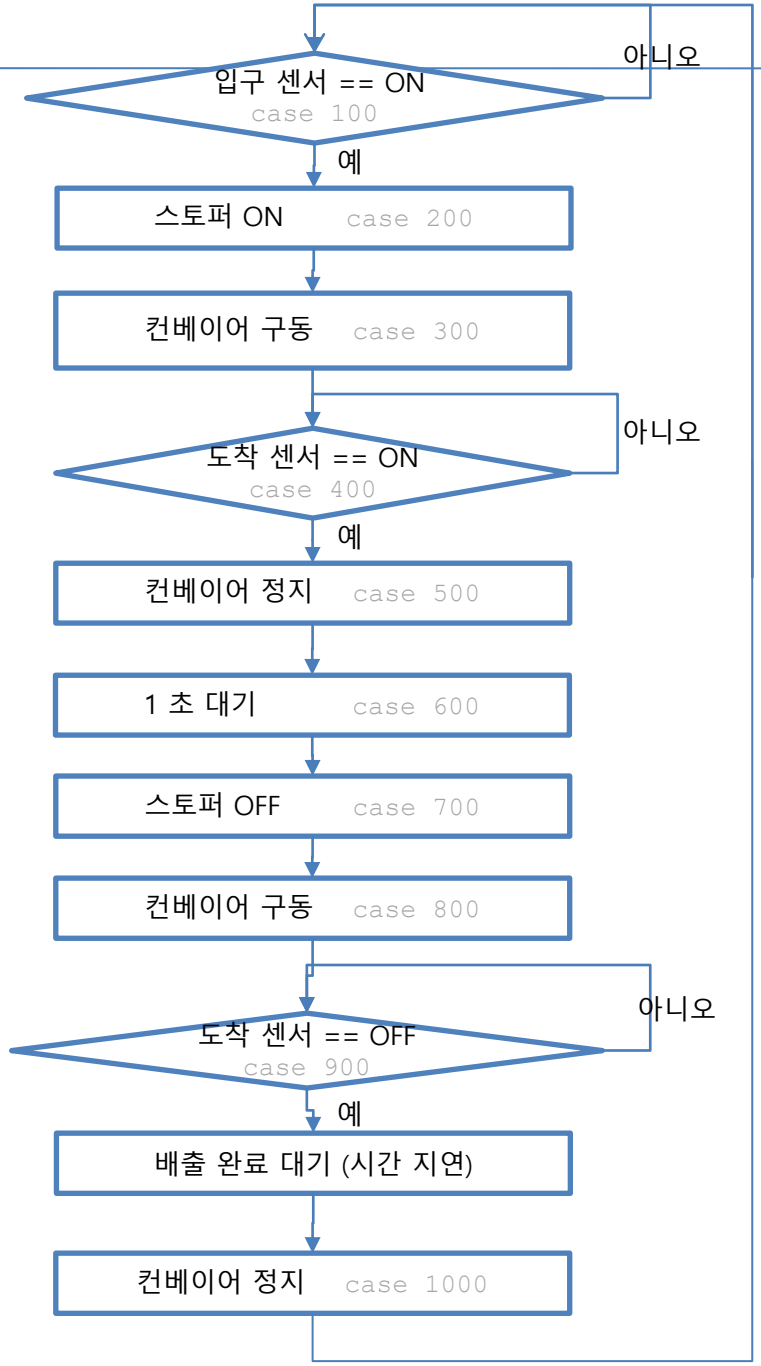




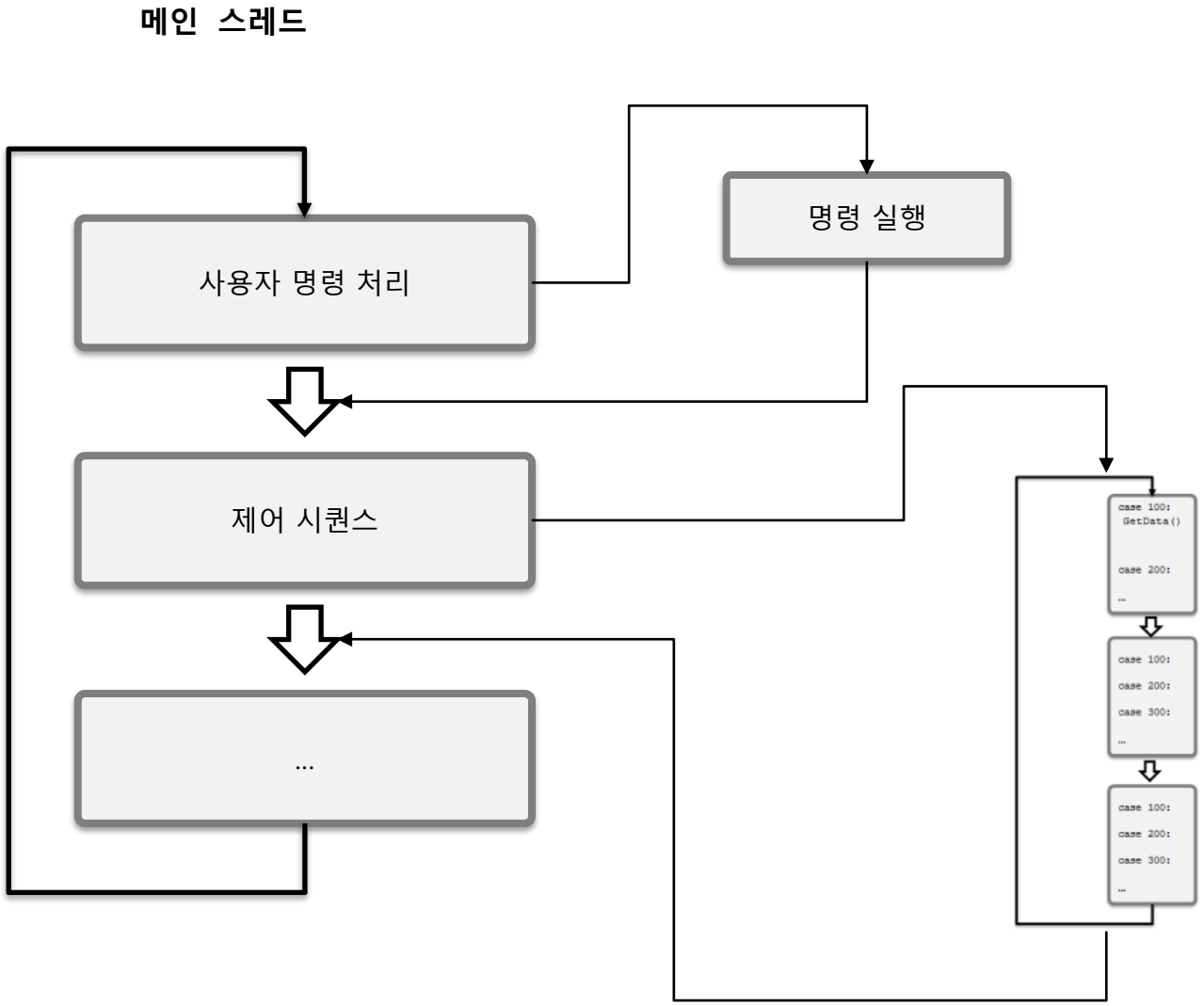
제어 시퀀스 작성 방법

while 반복문

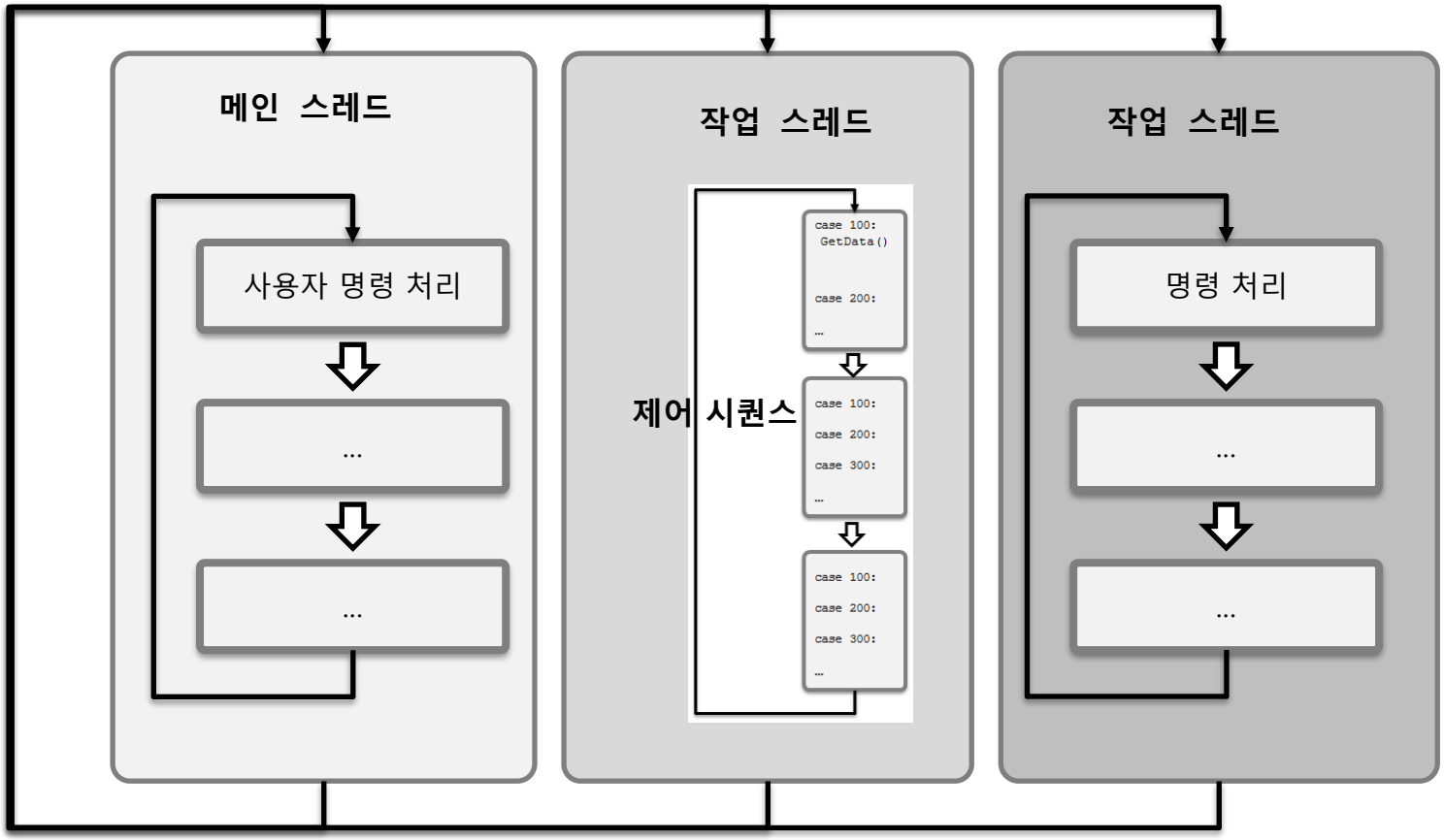




GUI(Main) 스레드에 제어 시퀀스 작성



멀티 스레드 프로그램의 동작 구조



스레드간 작업 전환 (콘텍스트 스위칭)

Thread 의 생성

```
Thread thread = new Thread(ReceiveMsgHandleThread);
thread.Start();
```

```
private void ReceiveMsgHandleThread()
{
    _bThreadStop = false;
    while (_bThreadStop == false)
    {
        ……

        Thread.Sleep(50);
    }
}
```

Conveyor 구동을 위한 IO Map (PCB Cleaner)

PLC --> PC			PLC <-- PC		
Adress		Description	Adress		Description
R000		에러 플래그	R100		에러 플래그 클리어
R001		-	R101		컨베어 정방향 구동
R002		-	R102		컨베어 역방향 구동
R003		-	R103		컨베어 정지
R004		컨베어 배출위치 이동완료	R104		컨베어 배출위치 이동

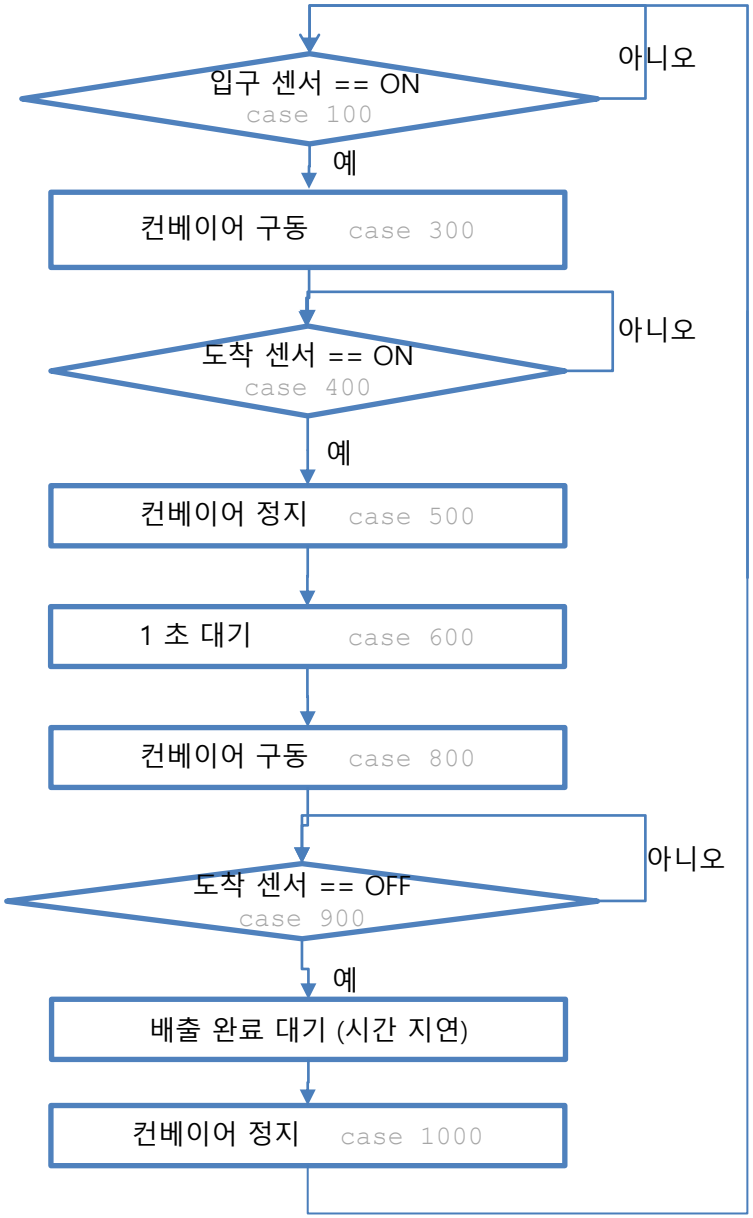
Out sensor 위치에 정지

PLC <-- PC		
Adress		Description
DT100		컨베어 가속시간 [ms단위]
DT101		컨베어 감속시간 [ms단위]
DT102		컨베어 구동 속도값 [하위 16bit]
DT103		컨베어 구동 속도값 [상위 16bit]

X000		Sensor: Conv' Width Home	Y000		
X001		Sensor: Conv' Width Over	Y001		
X002		Sensor: 'In Shuttle Open Position	Y002		
X003		Sensor: 'In Shuttle Close Position	Y003		
X004		Sensor: 'Out Shuttle Open Position	Y004		
X005		Sensor: 'Out Shuttle Close Position	Y005		
X006		Signal: Door Interlock	Y006		-
X007		Signal: Emergency Stop	Y007		-
X008		Sensor: Conveyor In	Y008		-
X009			Y009		In Shutter Open
X00A		Sensor: Lower Conveyor Slow	Y00A		In Shutter Close
X00B		Sensor: Lower Conveyor Out	Y00B		Out Shutter Open
X00C			Y00C		Out Shutter Close
X00D			Y00D		Air Blower On/Off
X00E		SMEMA Board Available N-1	Y00E		SMEMA Not Busy N-1
X00F		SMEMA Not Busy N+1	Y00F		SMEMA Board Available N+1

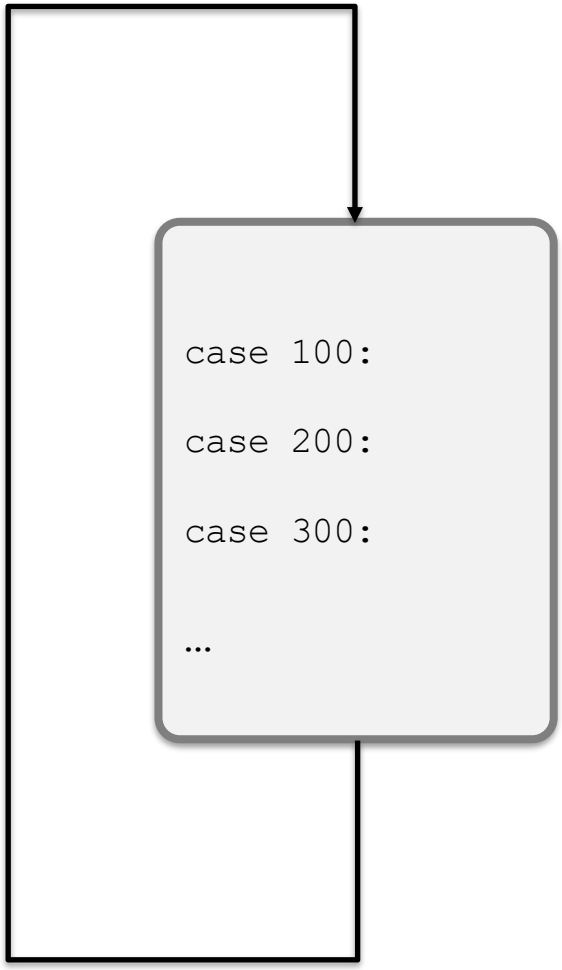
실습 2

1. 컨베이어 제어 하기



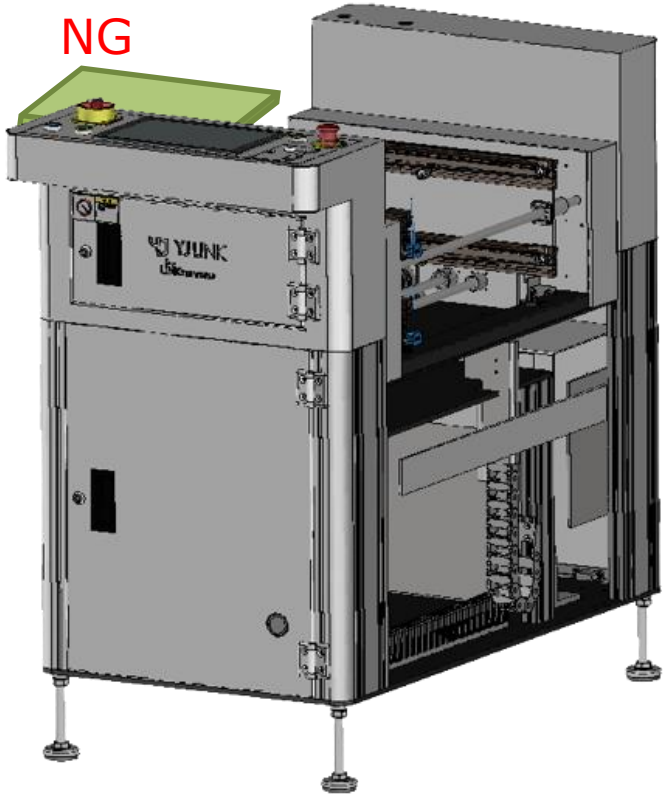
함수 호출에 의한 동작 제어

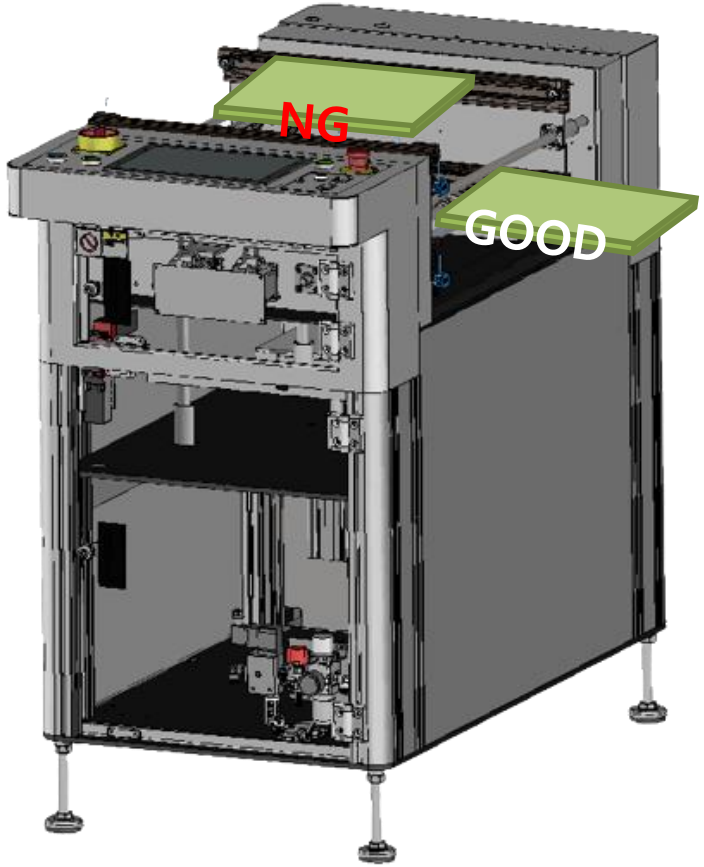
while 반복문




```
case 400:
  if(OFF == GetData(LOADER_STOP_SENSOR))
  {
     Sleep(2000);
     SetData(LOADER_MOTOR, OFF);
     StepNum = 100;
  }
  break;
```

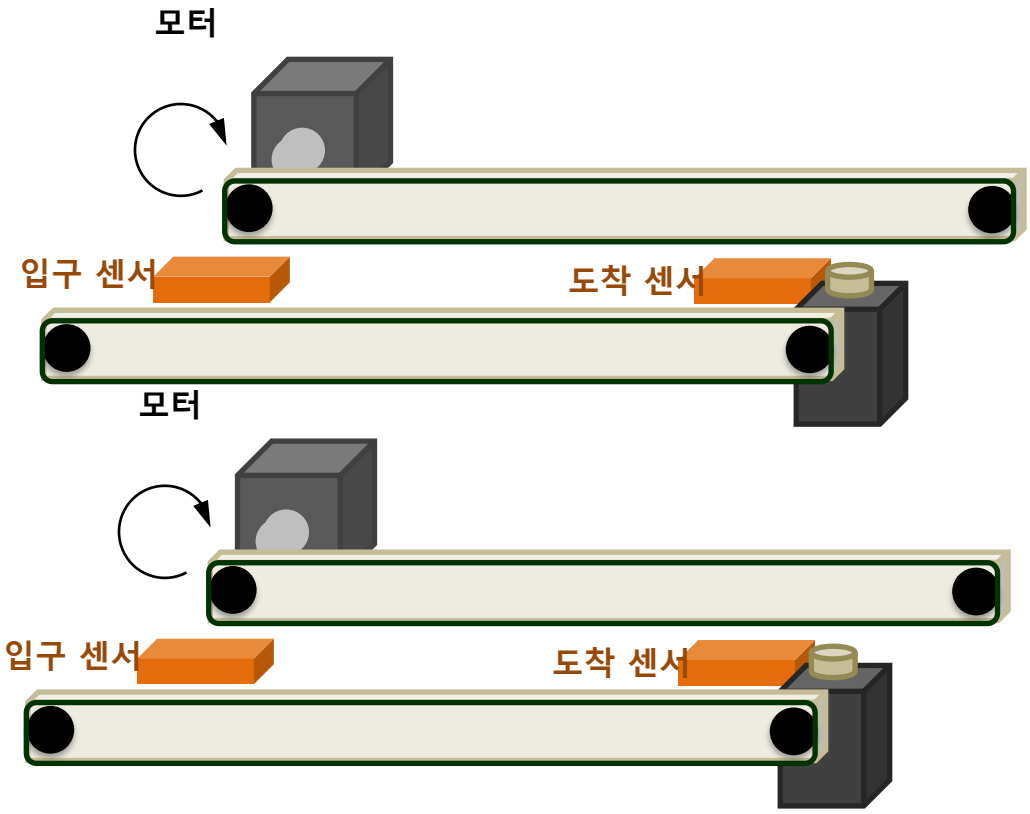
Reject Conveyor 동작 설명





실습 3

1. 실습 2에서 작성한 코드에서 Reject Conveyor 에서와 같이 컨베이어가 하나 더 있을 경우의 코드 작성

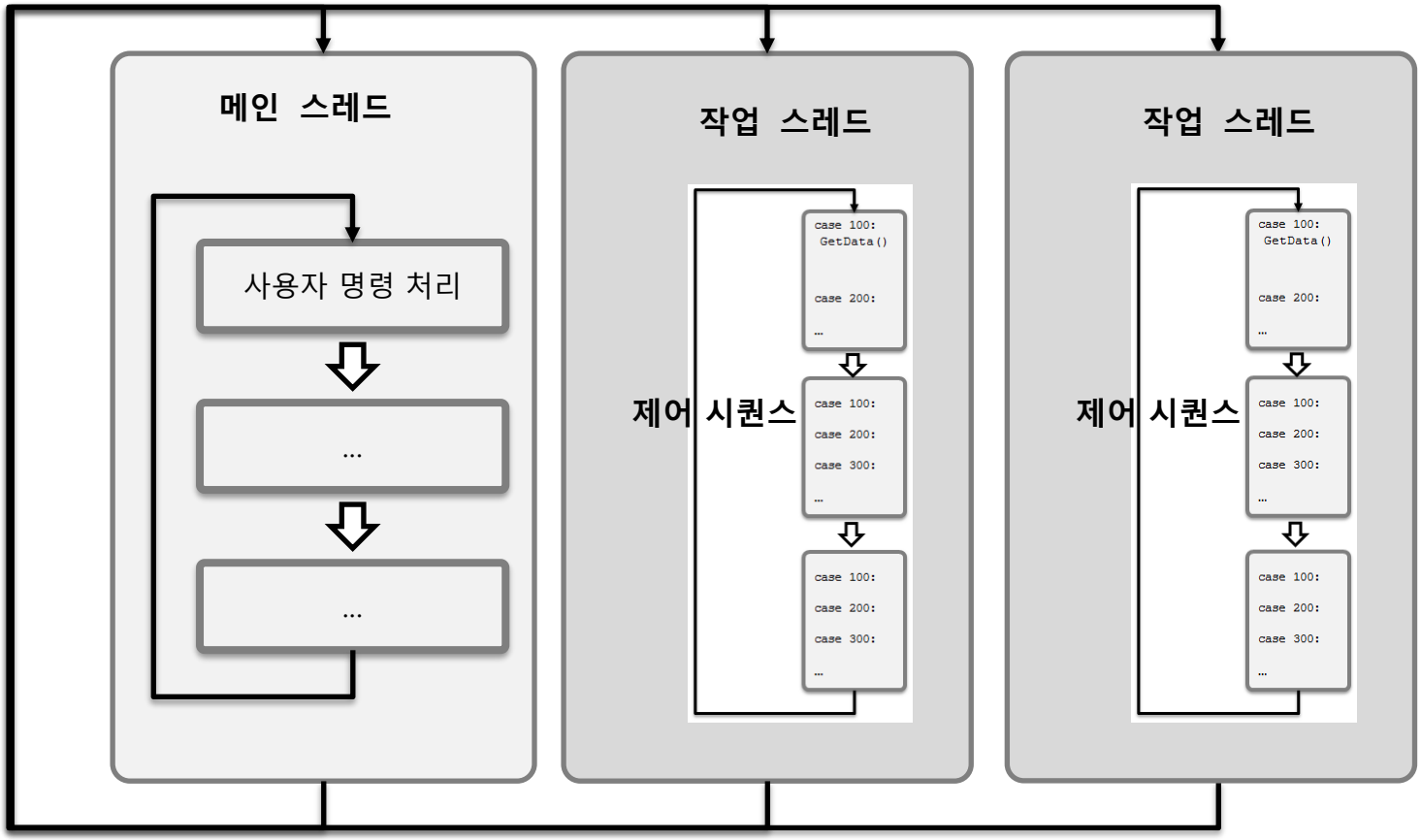


실습 3

1. 컨베이어가 하나 더 있을 경우의 코드 작성

실습 3

1. 컨베이어가 하나 더 있을 경우의 코드 작성

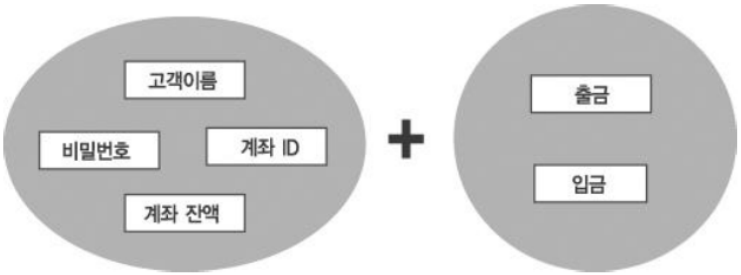


실습 3

1. 컨베이어가 하나 더 있을 경우의 코드 작성

객체 지향 프로그래밍

구조체 + 함수



- 함수를 넣으면 좋은 구조체
  - 프로그램=데이터+데이터 조작 루틴(함수)
  - 잘 구성된 프로그램은 데이터와 더불어 함수들도 그룹화
  - 객체지향 프로그래밍 기법

- 객체지향 프로그래밍에서는 프로그램은 여러 개의 객체로 구성
  - 객체(object)는 자료(특성(attribute))와 이를 대상으로 처리하는 동작인 연산(함수,메쏘드(method))을 하나로 묶어 만든 요소로 프로그램을 구성하는 실체
  - 객체란 단순히 자료를 표현하는 변수 만을 가지는 것이 아니라 그 객체가 무엇을 할 수 있는가를 정의한 함수(메쏘드)로 구성

객체 지향 프로그래밍 – 용어 정리

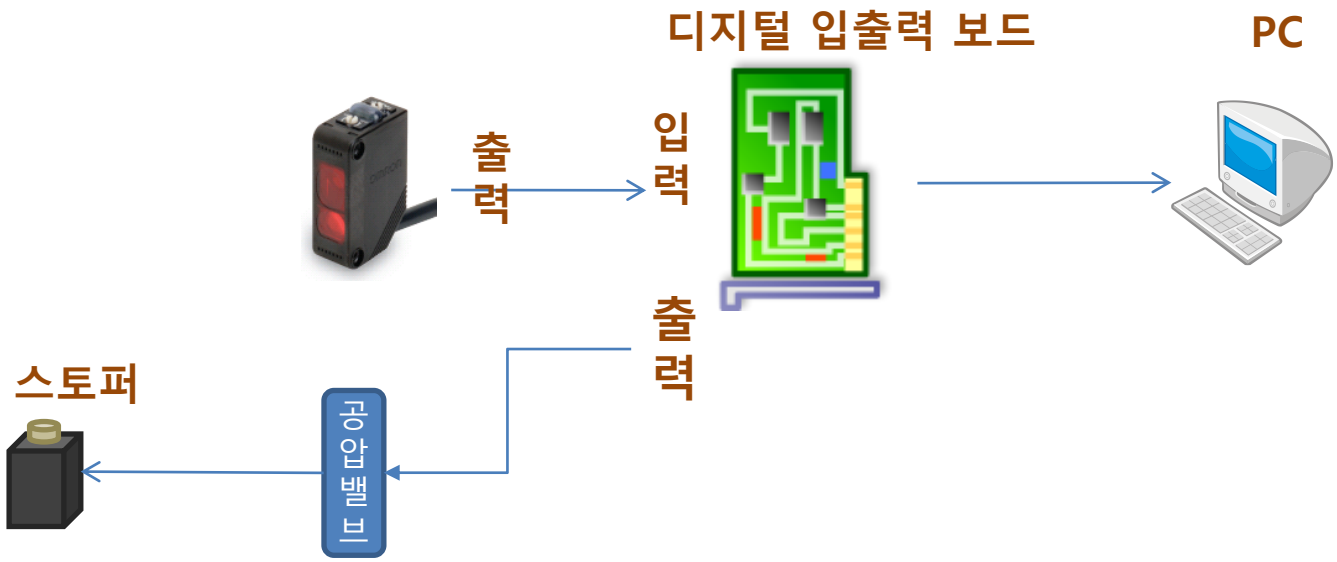
```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

왼쪽의 Car 클래스를 대상으로 생성된 변수를 가리켜 '객체'라 한다.

왼쪽의 Car 클래스 내에 선언된 변수를 가리켜 '멤버변수'라 한다.

왼쪽의 Car 클래스 내에 정의된 함수를 가리켜 '멤버함수'라 한다.

CDIO 객체를 이용한 Conveyor Class



CDIO 클래스

함수 호출과 CDIO 클래스의 사용 코드의 비교

CConveyor 클래스 다이어그램

작업 스테이지에서의 CConveyor 객체 생성

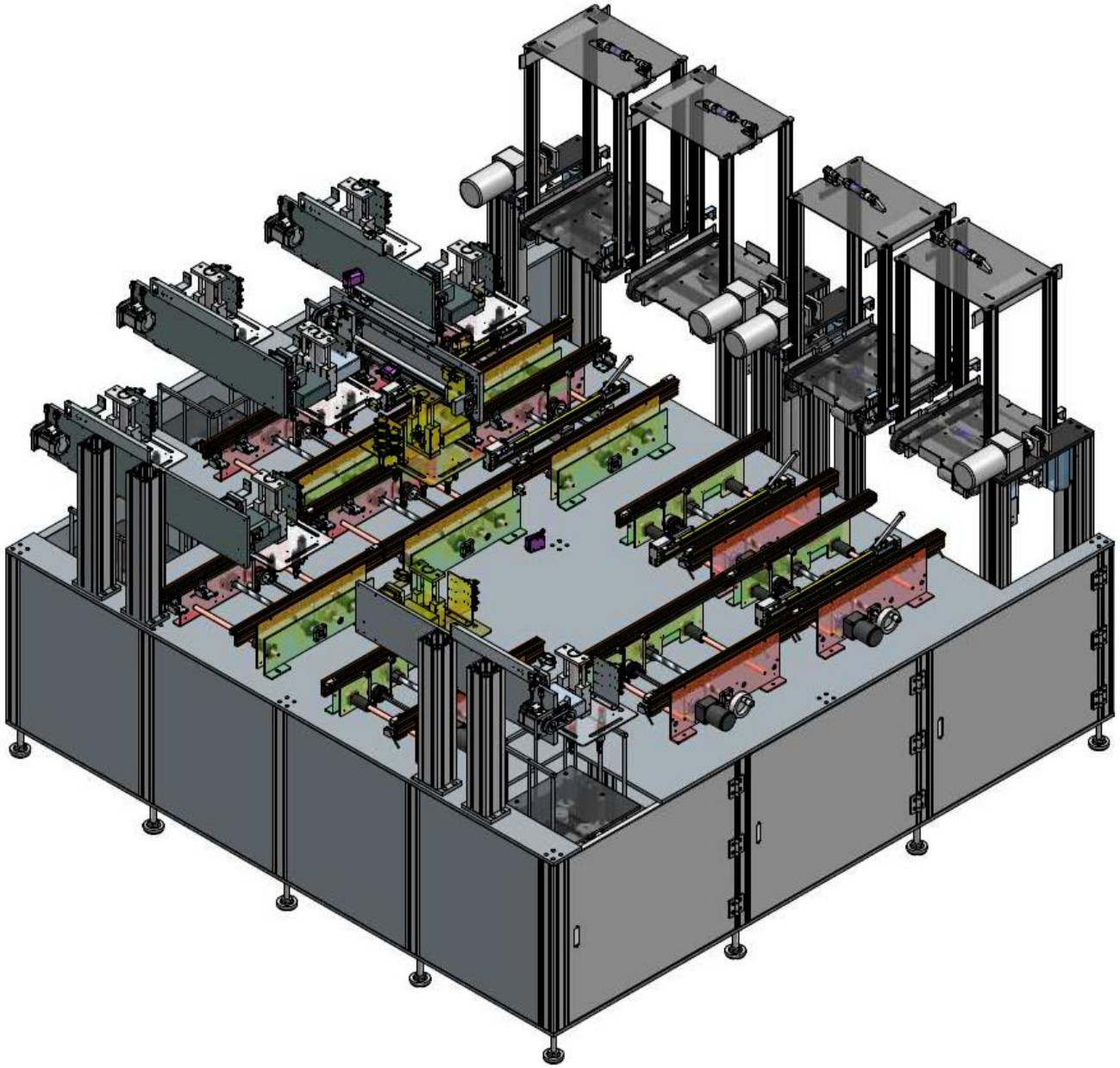
실습 4

1. 실습 3에서 작성한 코드를 객체 지향 코드로 변경해서 작성

실습 4

1. 실습 3에서 작성한 코드를 객체 지향 코드로 변경해서 작성

설비 예시



객체 지향 프로그래밍 – 용어 정리

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

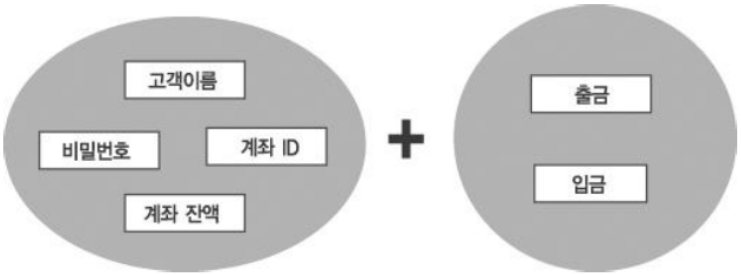
왼쪽의 Car 클래스를 대상으로 생성된 변수를 가리켜 '객체'라 한다.

왼쪽의 Car 클래스 내에 선언된 변수를 가리켜 '멤버변수'라 한다.

왼쪽의 Car 클래스 내에 정의된 함수를 가리켜 '멤버함수'라 한다.

객체 지향 프로그래밍

구조체 + 함수



- 함수를 넣으면 좋은 구조체
  - 프로그램=데이터+데이터 조작 루틴(함수)
  - 잘 구성된 프로그램은 데이터와 더불어 함수들도 그룹화
  - 객체지향 프로그래밍 기법

- 객체지향 프로그래밍에서는 프로그램은 여러 개의 객체로 구성
  - 객체(object)는 자료(특성(attribute))와 이를 대상으로 처리하는 동작인 연산(함수,메쏘드(method))을 하나로 묶어 만든 요소로 프로그램을 구성하는 실체
  - 객체란 단순히 자료를 표현하는 변수 만을 가지는 것이 아니라 그 객체가 무엇을 할 수 있는가를 정의한 함수(메쏘드)로 구성

다형성 1

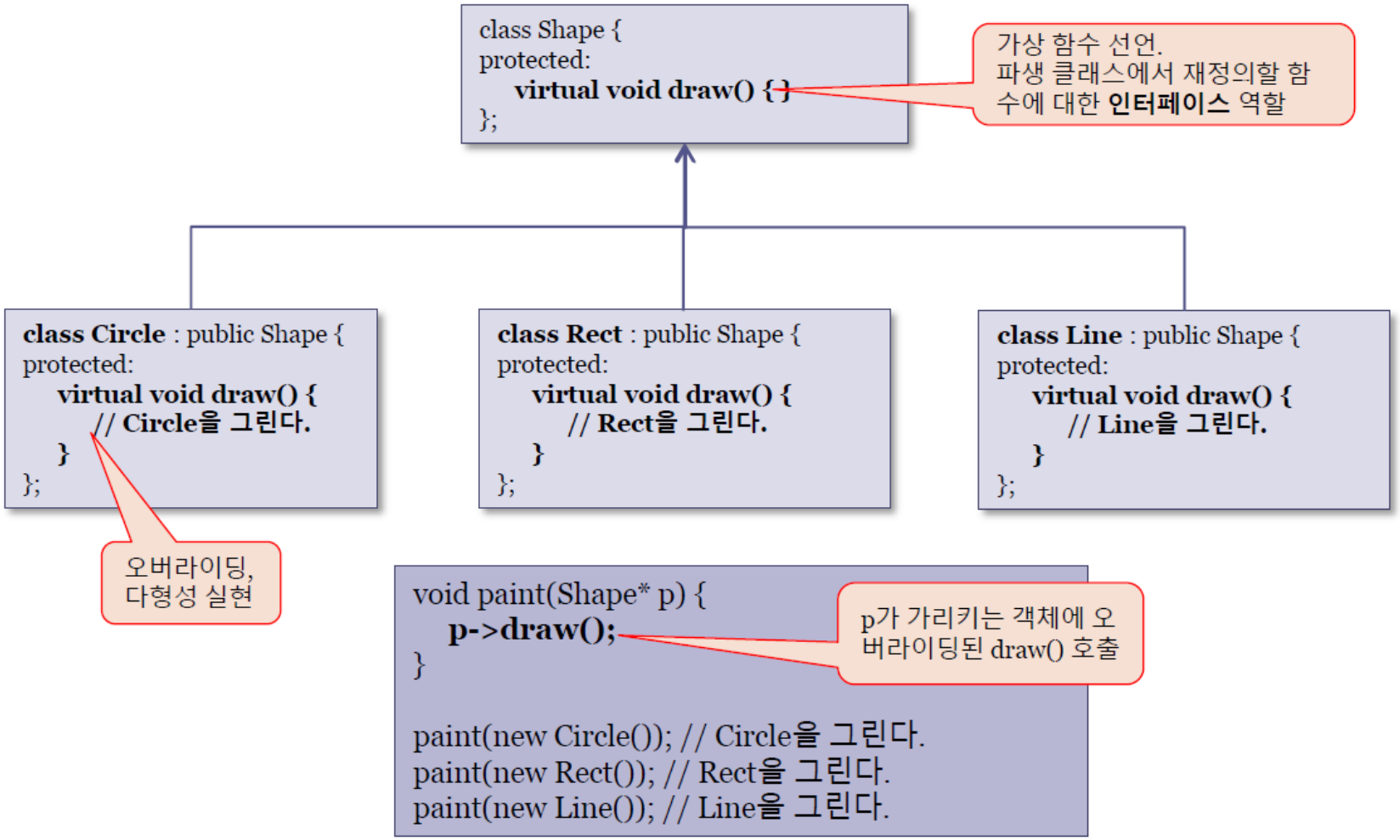
- 다형성이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것

- 다형성은 객체 지향 기법에서 하나의 코드로 다양한 타입의 객체를 처리하는 중요한 기술

- 다형성은 동질이상의 의미
  - 모습은 같은데 형태는 다르다
  - 문장은 같은데 결과는 다르다

다형성 2

- draw() 가상 함수를 가진 기본 클래스 Shape
- 오버라이딩을 통해 Circle, Rect, Line 클래스에서 자신만의 draw() 구현 (➔ 뒷장 연결)



다형성 – 재 정의 함수의 호출

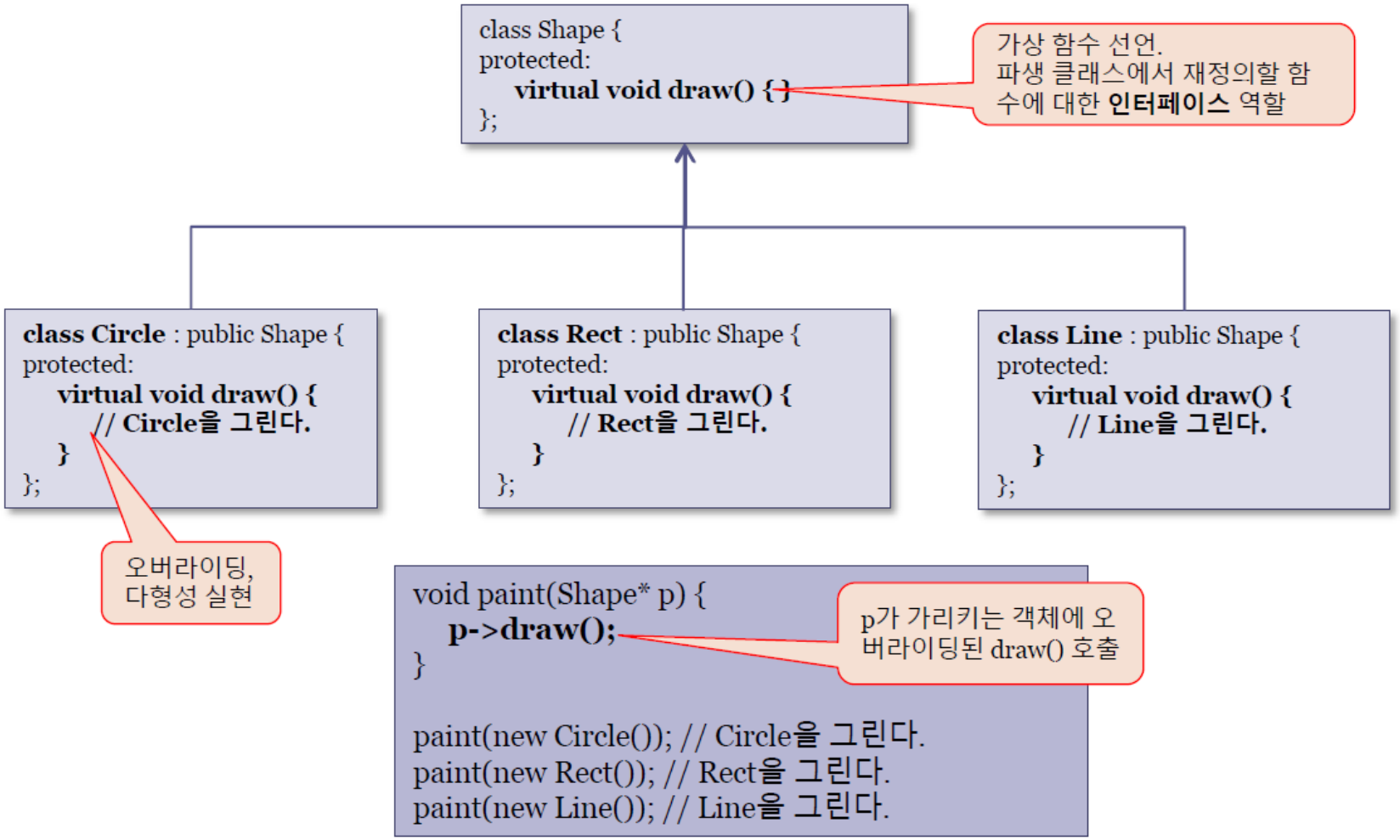
CMotor 추상 클래스

다형성을 지원하는 모터 클래스 구조

추상 클래스와 구현 클래스와의 관계

다형성 2

- draw() 가상 함수를 가진 기본 클래스 Shape
- 오버라이딩을 통해 Circle, Rect, Line 클래스에서 자신만의 draw() 구현 (➔ 뒷장 연결)



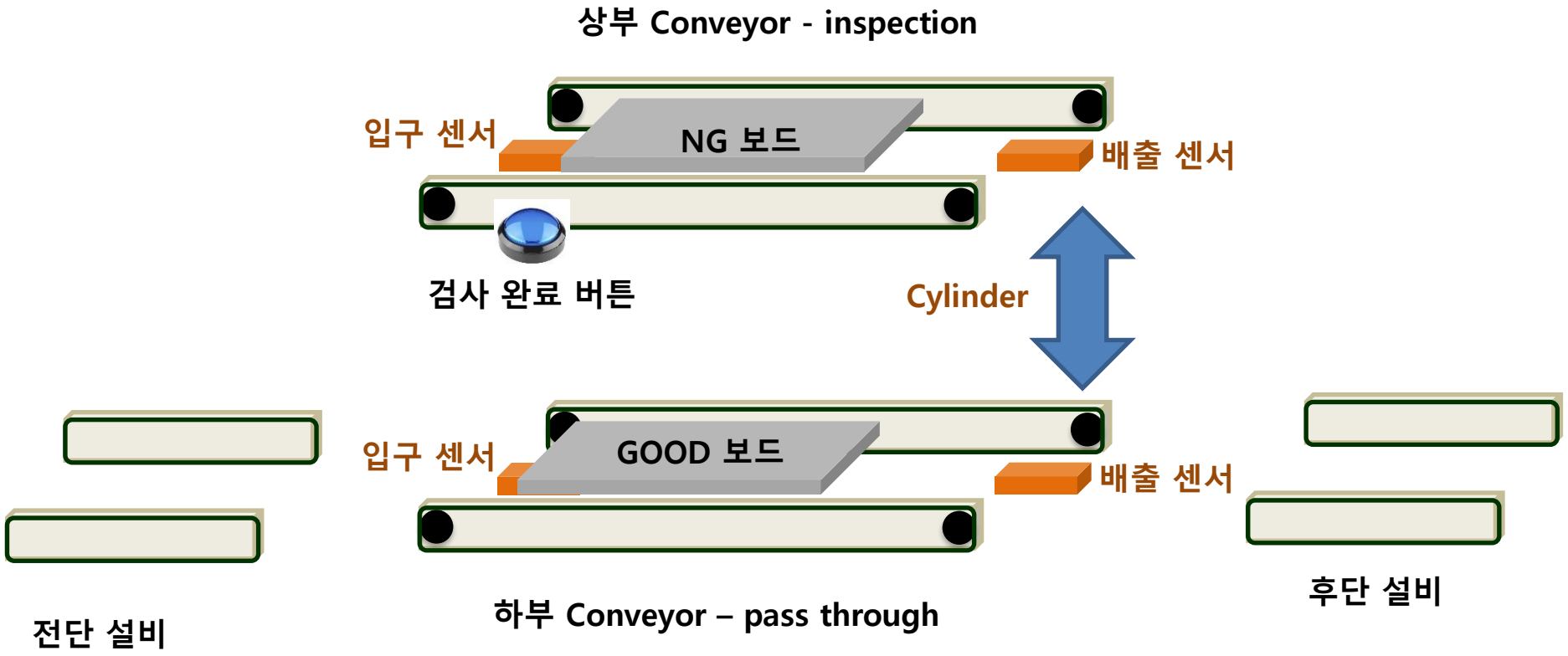
다형성 1

- 다형성이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것

- 다형성은 객체 지향 기법에서 하나의 코드로 다양한 타입의 객체를 처리하는 중요한 기술

- 다형성은 동질이상의 의미
  - 모습은 같은데 형태는 다르다
  - 문장은 같은데 결과는 다르다

Reject Conveyor 동작 설명



실습 5

1. Reject Conveyor 구현

Rejecter 의 CheckBaAndOtherCheck 함수

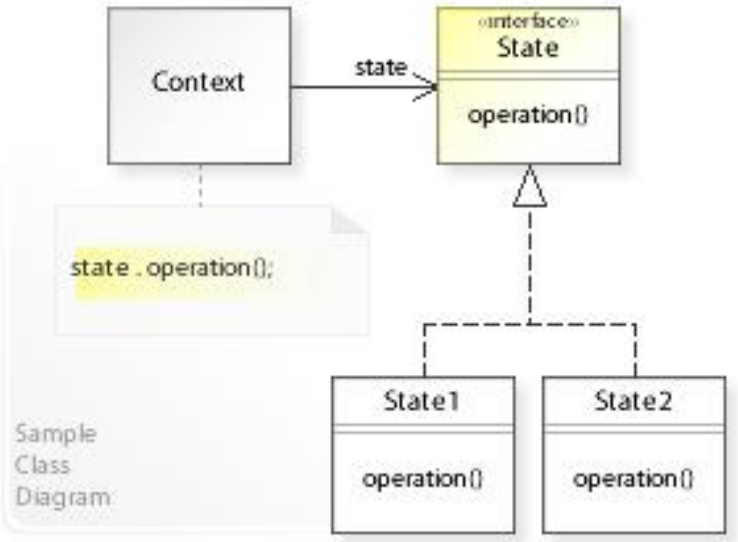
Reject Conveyor 의 상태 변화

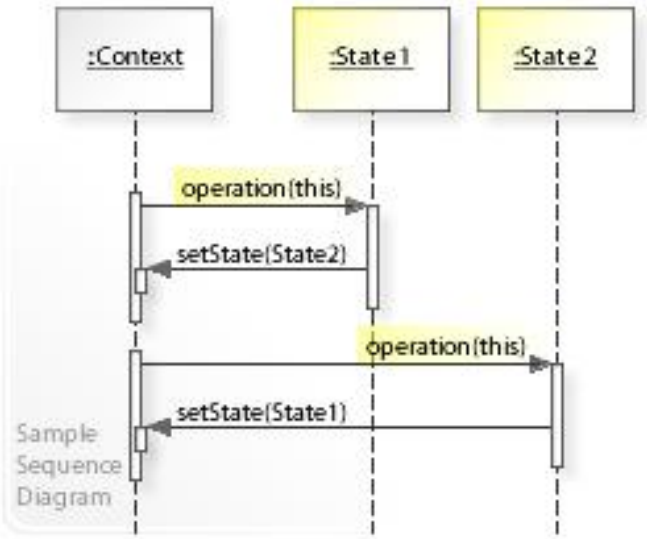
State 별 클래스 정의

Rejecter 의 Auto Sequence

Rejecter 의 Auto Sequence 전 설비 상태 확인

State Pattern

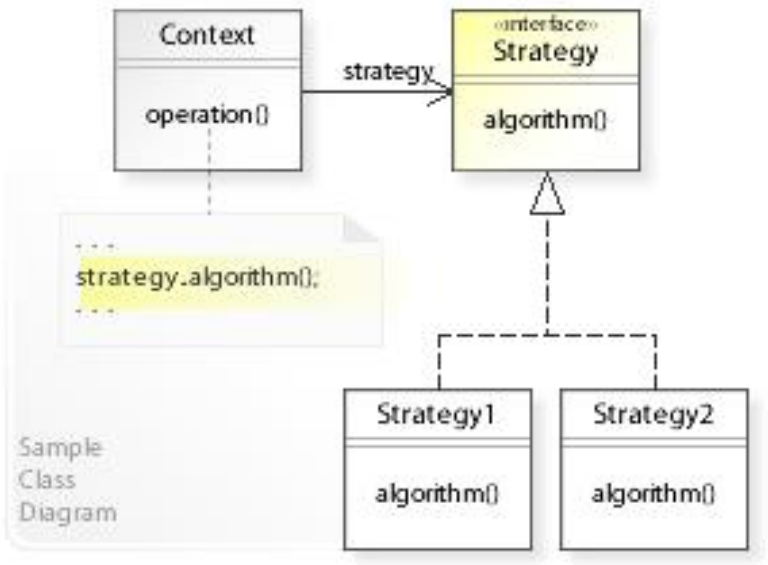


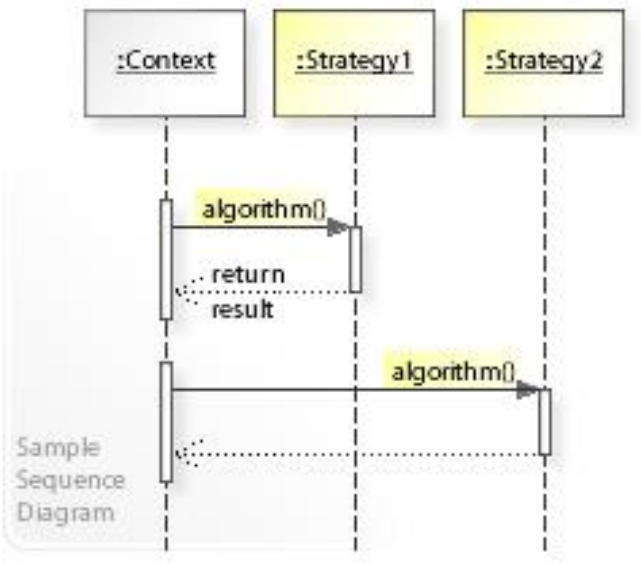


상태 패턴(state pattern)은 객체 지향 방식으로 상태 기계를 구현하는 행위 소프트웨어 디자인 패턴이다. 상태 패턴을 이용하면 상태 패턴 인터페이스의 파생 클래스로서 각각의 상태를 구현함으로써, 또 패턴의 슈퍼클래스에 의해 정의되는 메소드를 호출하여 상태 변화를 구현함으로써 상태 기계를 구현한다.

상태 패턴은 패턴의 인터페이스에 정의된 메소드들의 호출을 통해 현재의 전략을 전환할 수 있는 전략 패턴으로 해석할 수 있다.

Strategy Pattern 정의



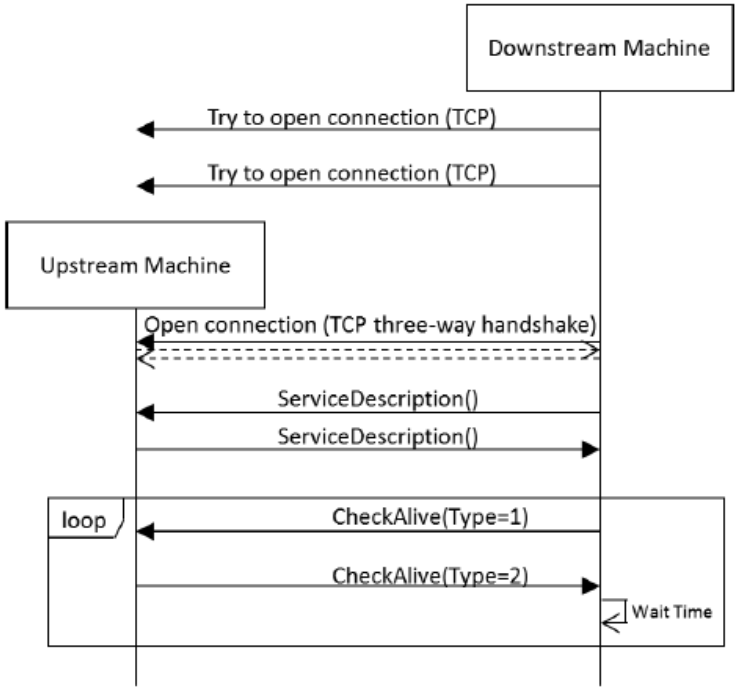


전략 패턴(strategy pattern) 또는 **정책 패턴**(policy pattern)은 실행 중에 알고리즘을 선택할 수 있게 하는 행위 소프트웨어 디자인 패턴이다. 전략 패턴은

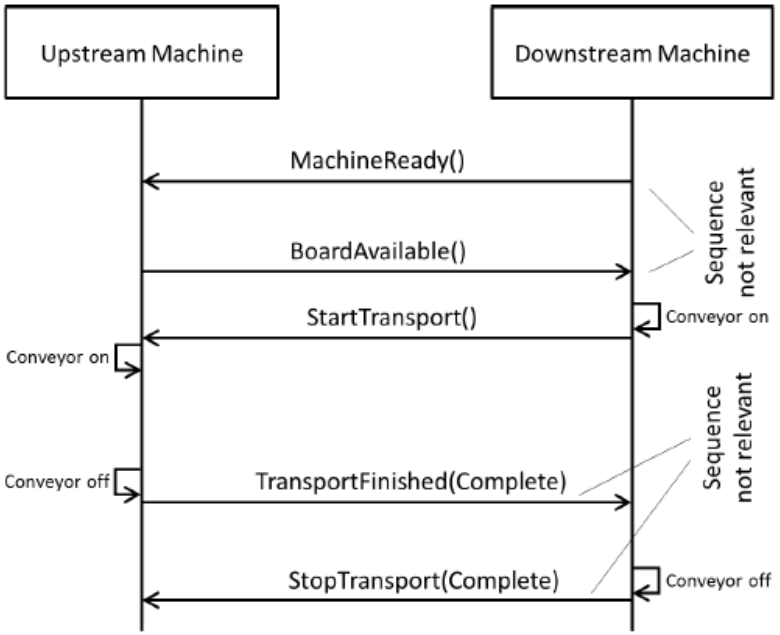
- 특정한 계열의 알고리즘들을 정의하고
- 각 알고리즘을 캡슐화하며
- 이 알고리즘들을 해당 계열 안에서 상호 교체가 가능하게 만든다.

전략은 알고리즘을 사용하는 클라이언트와는 독립적으로 다양하게 만든다.[1] 전략은 유연하고 재사용 가능한 객체 지향 소프트웨어를 어떻게 설계하는지 기술하기 위해 디자인 패턴의 개념을 보급시킨 디자인 패턴(Gamma 등)이라는 영향력 있는 책에 포함된 패턴들 가운데 하나이다.

Hermes 통신



Connection Sequence



Comm. Sequence

Hermes 통신

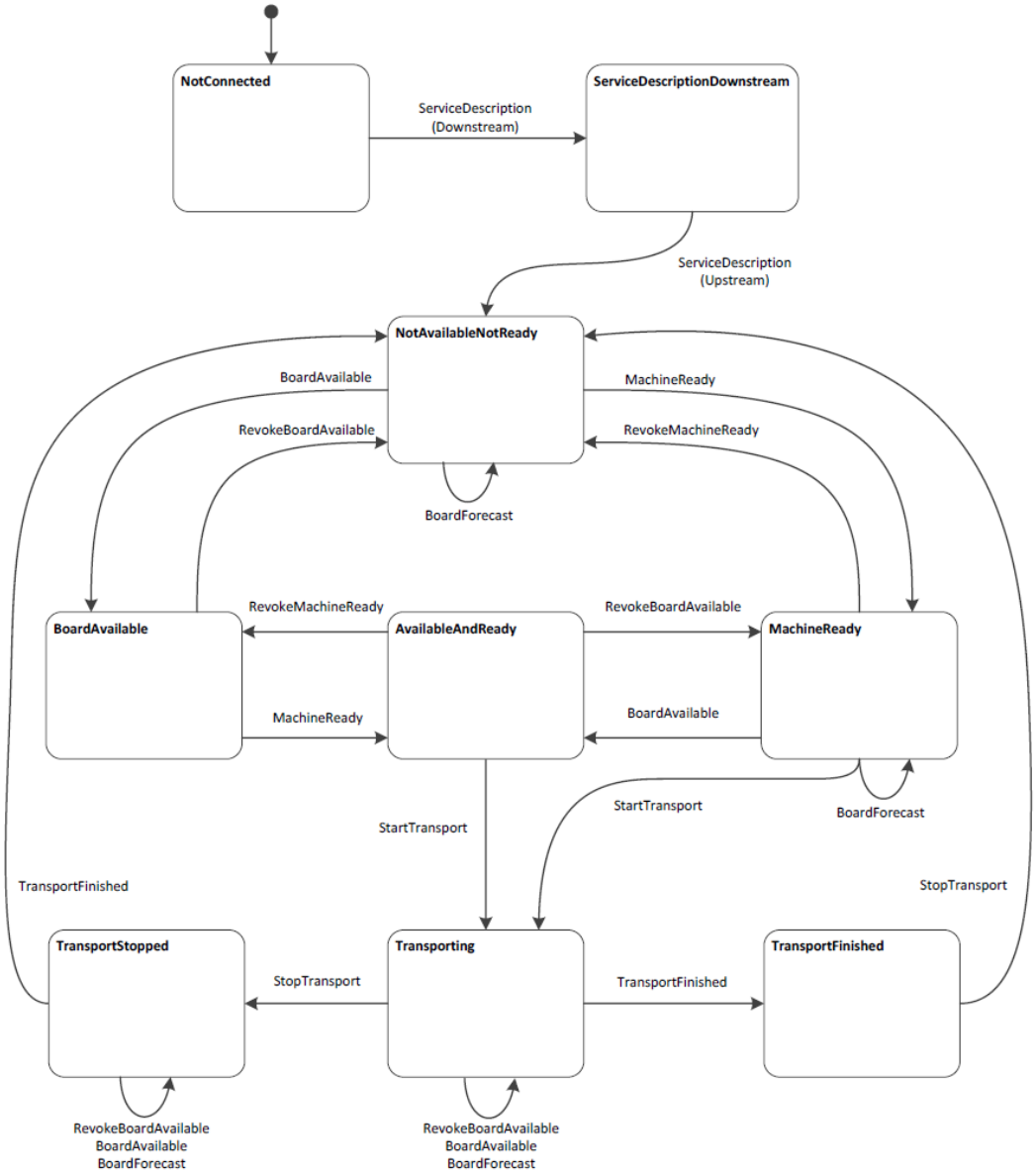


Fig. 19 Hermes interface states on horizontal channel