C언어 응용
전문가를 위한 C++

CHAPTER 1 C++ 부딪혀 보기

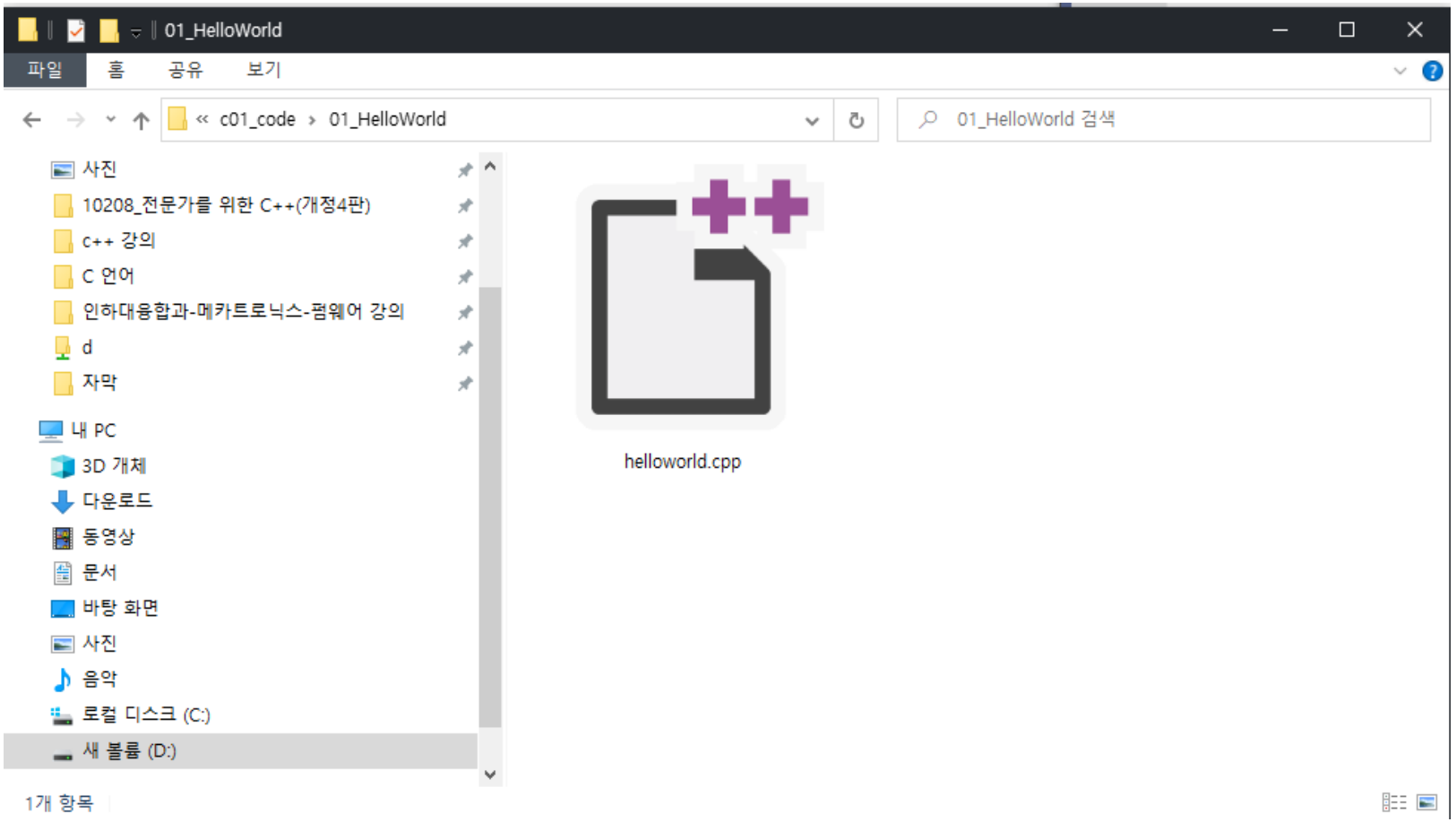
인하대학교
소프트웨어융합공학
가명현

Contents

- 들어가기 전에
- C++ 기초
- C++ 고급기능

들어가기 전에

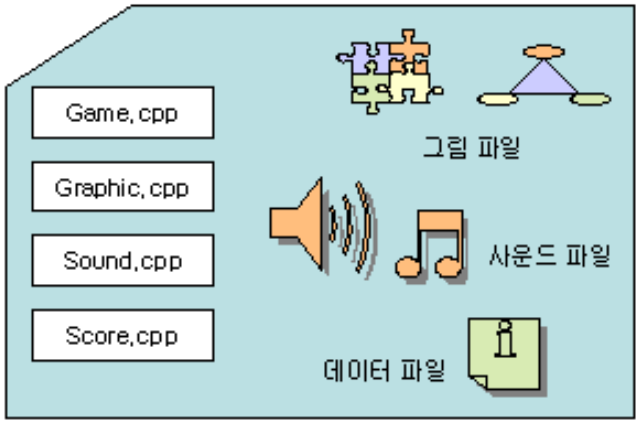
CPP 파일 실행하기



들어가기 전에

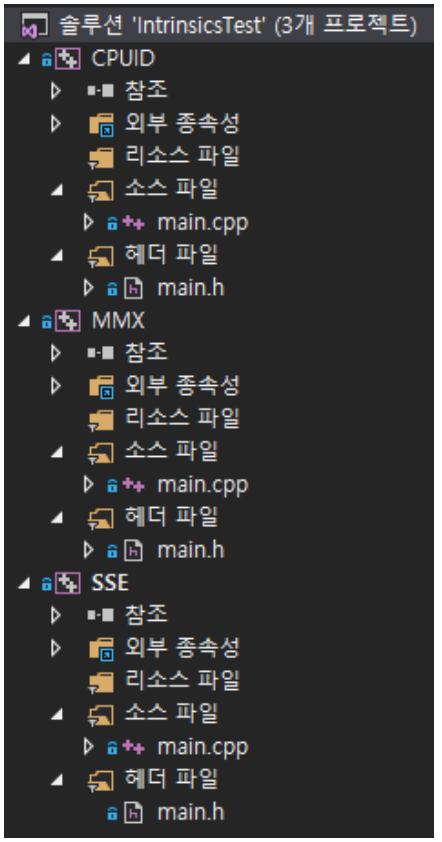
비주얼 스튜디오는 실무용 개발도구

- ✓ 말인즉슨, 친절하지 않다.
- ✓ Hello World 하나 실행시키기 자고 솔루션도 만들고 프로젝트도 만들어야 한다.
- ✓ 솔루션?? 프로젝트??
- ✓ 프로젝트 : 하나의 프로그램을 만드는 묶음
- ✓ 솔루션 : 하나 이상의 프로젝트가 모인 집합



프로젝트
≒ 파일
≒프로그램

솔루션



프로젝트와 솔루션(Visual Studio)

들어가기 전에

빌드??

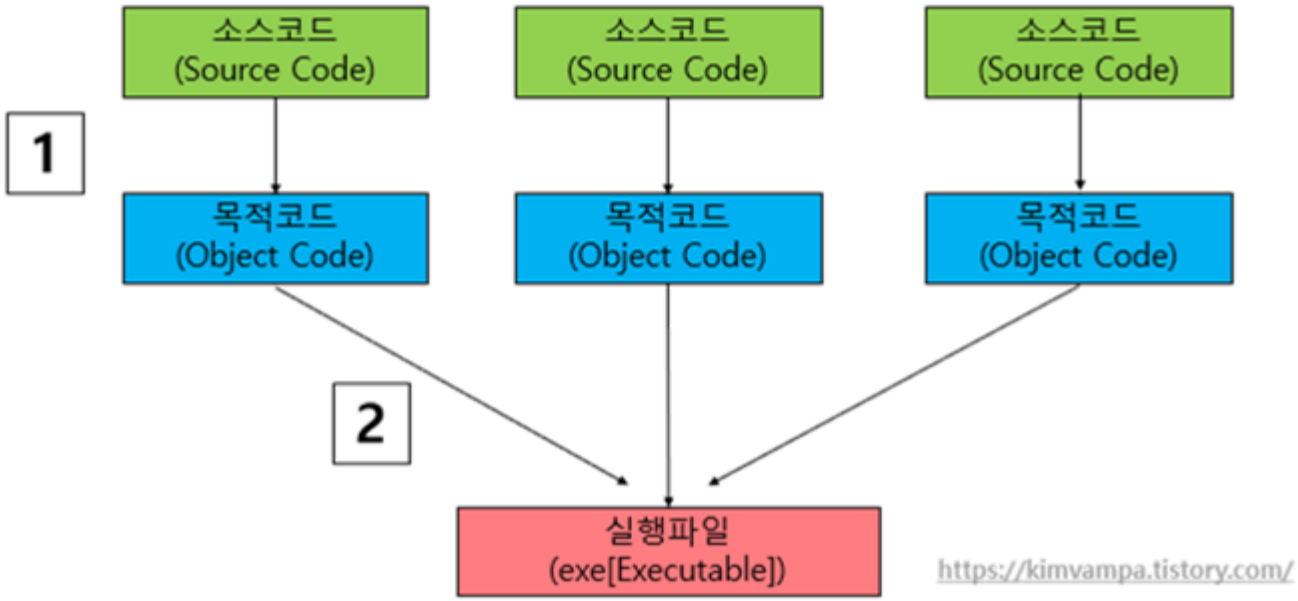
- ✓ 실행파일을 만드는 것
- ✓ 컴파일 과정과 링크 과정을 거쳐야 한다.

컴파일

- ✓ 소스코드를 기계어 파일로 만드는 과정

링크

- ✓ 컴파일된 파일들을 연결, 병합 해주는 과정



들어가기 전에

어셈블??

```
int i;
i = 35 + 56;
```

C++

컴파일

```
LOAD AX 35
LOAD BX 56
ADD AX BX
```

어셈블리어

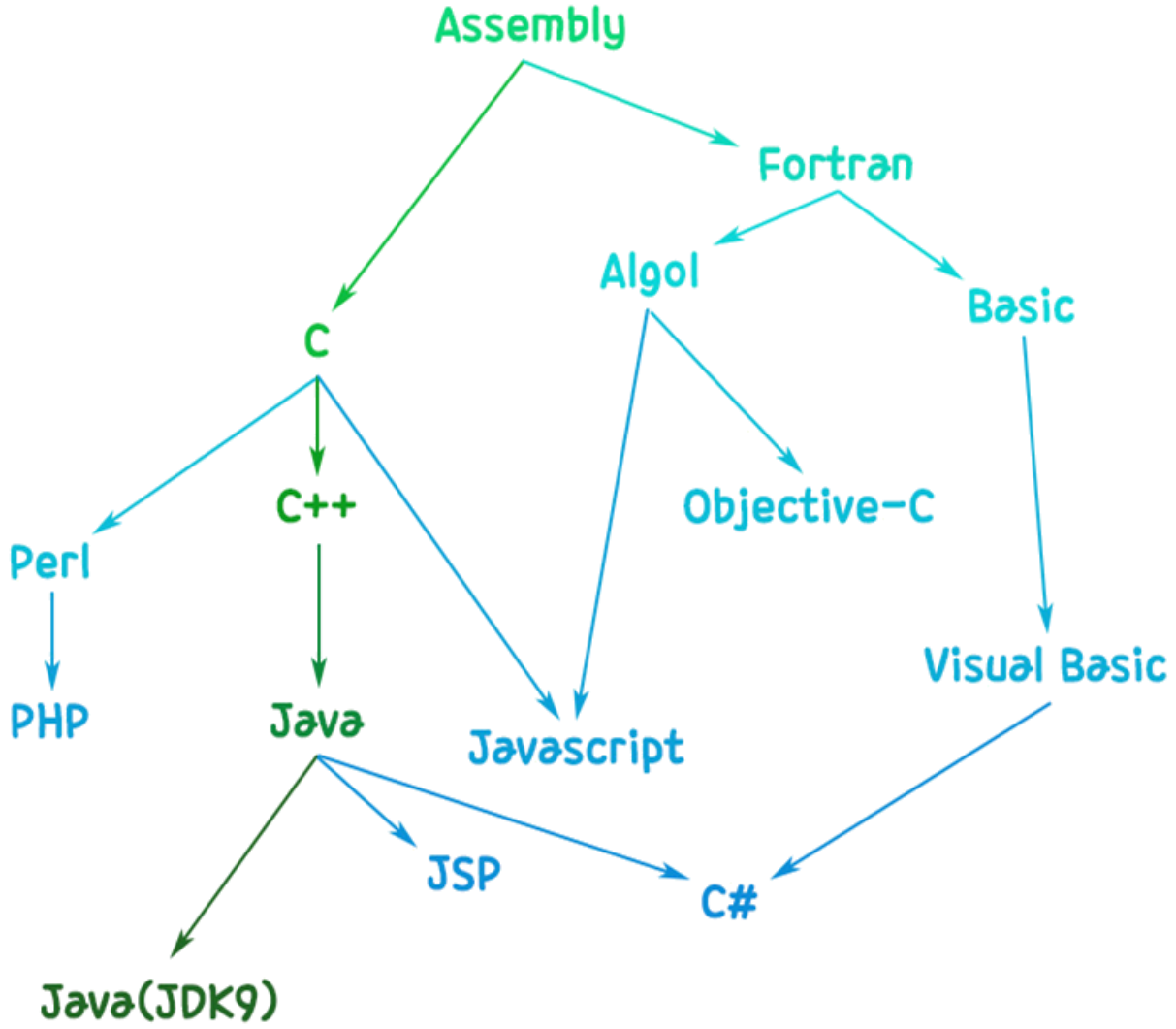
어셈블


```
01010000
00101001
11101011
```

기계어

들어가기 전에

프로그램 언어의 진화 과정

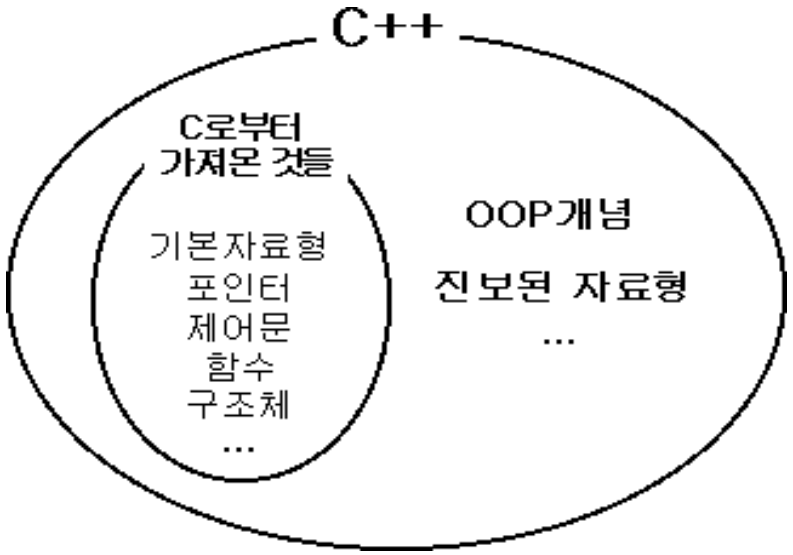


<명품 JAVA Programing에서 발췌>

들어가기 전에

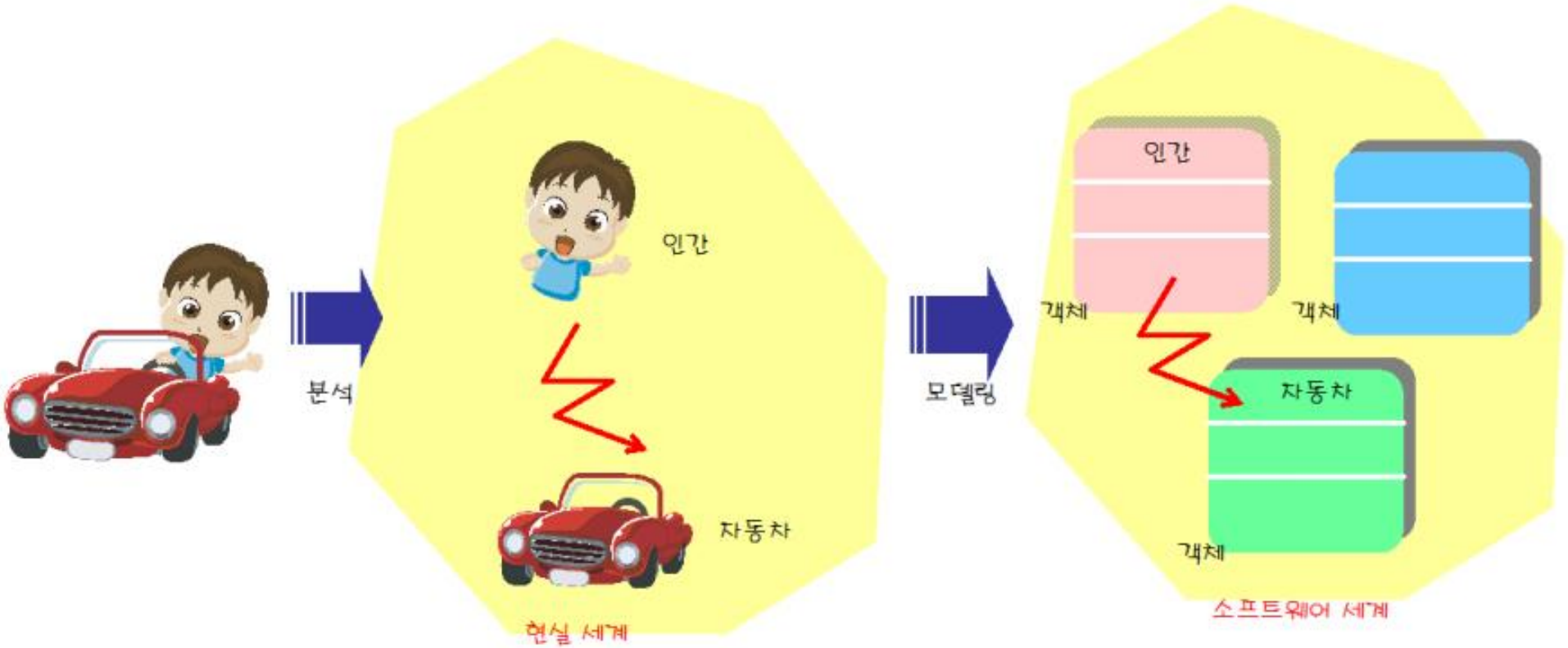
절차지향, 객체지향

절차지향
(Procedural Oriented Programming)
↓
객체지향
(Object Oriented Programming)

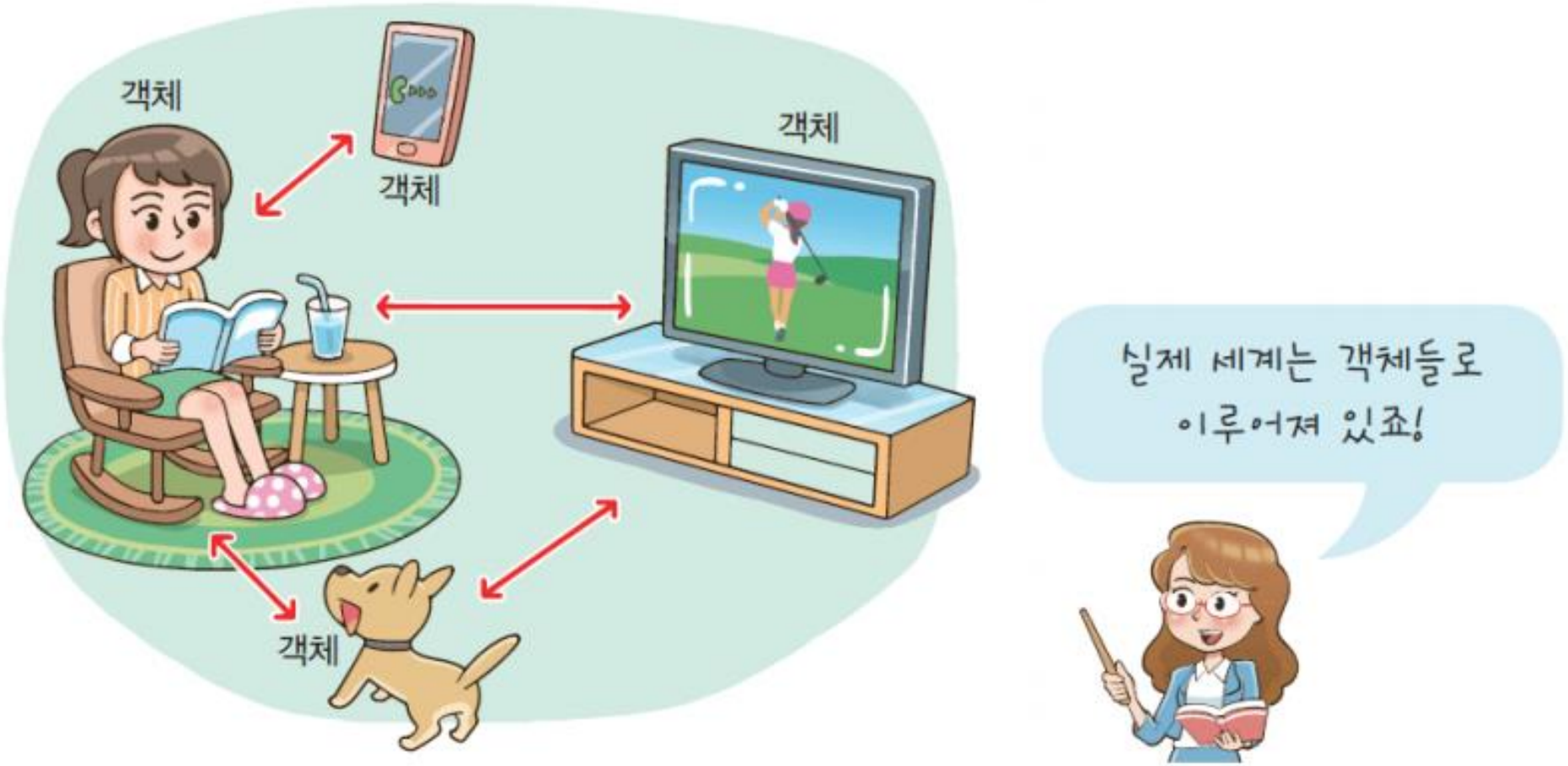


들어가기 전에

절차지향, 객체지향

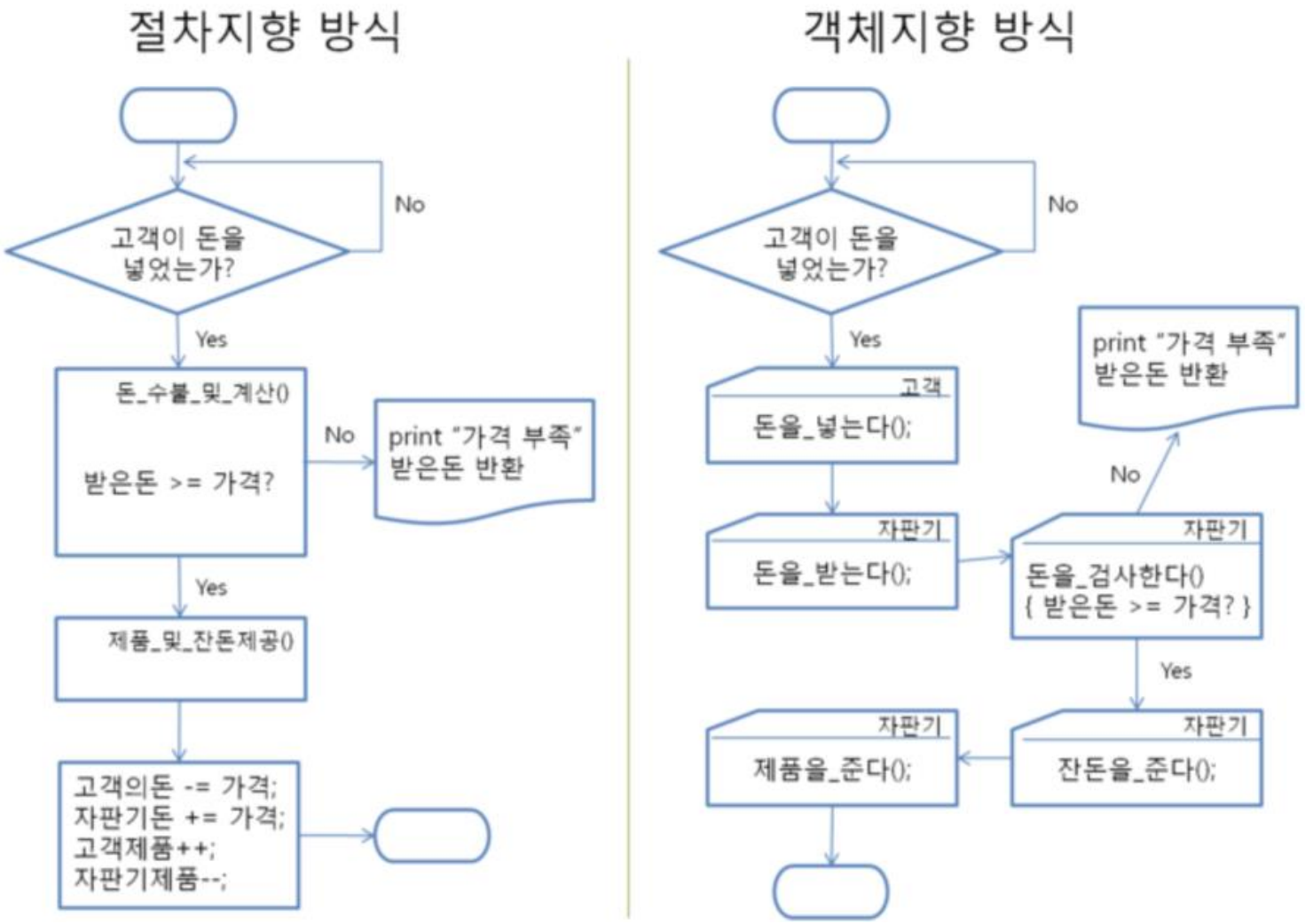


절차지향, 객체지향



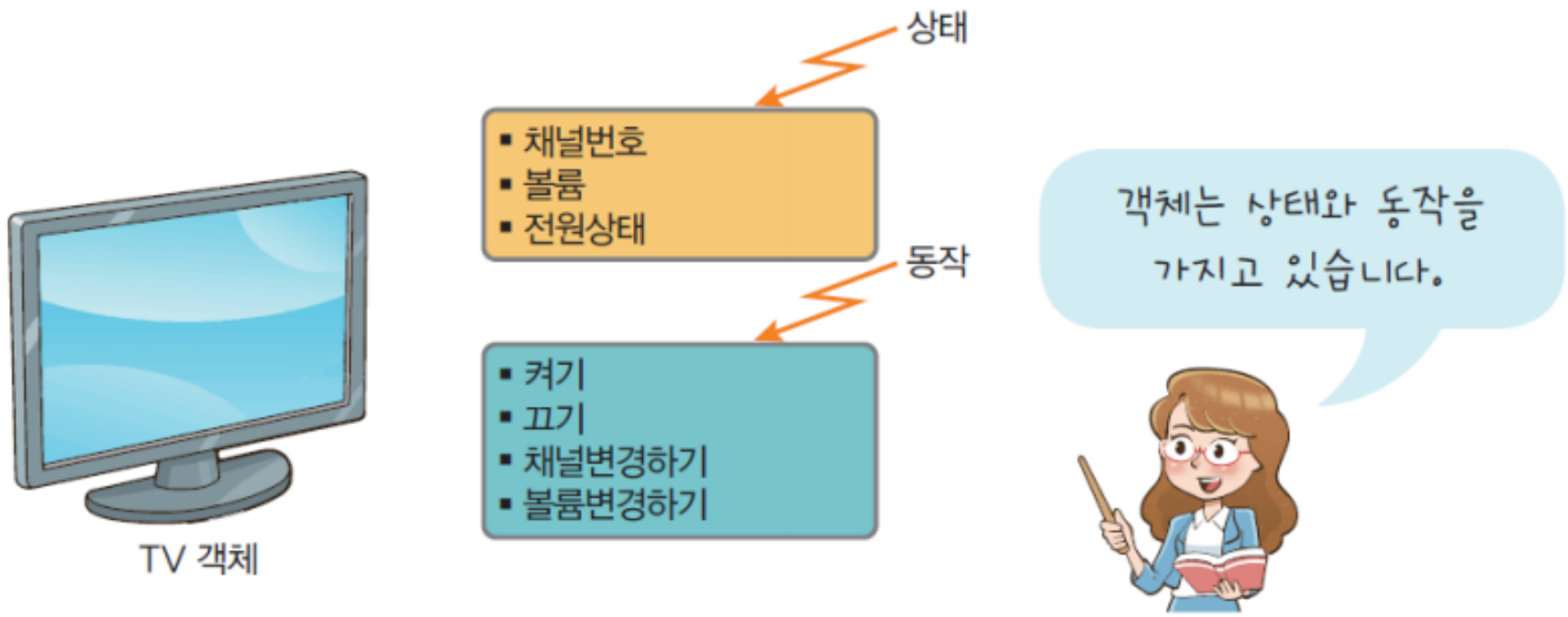
들어가기 전에

절차지향, 객체지향



들어가기 전에

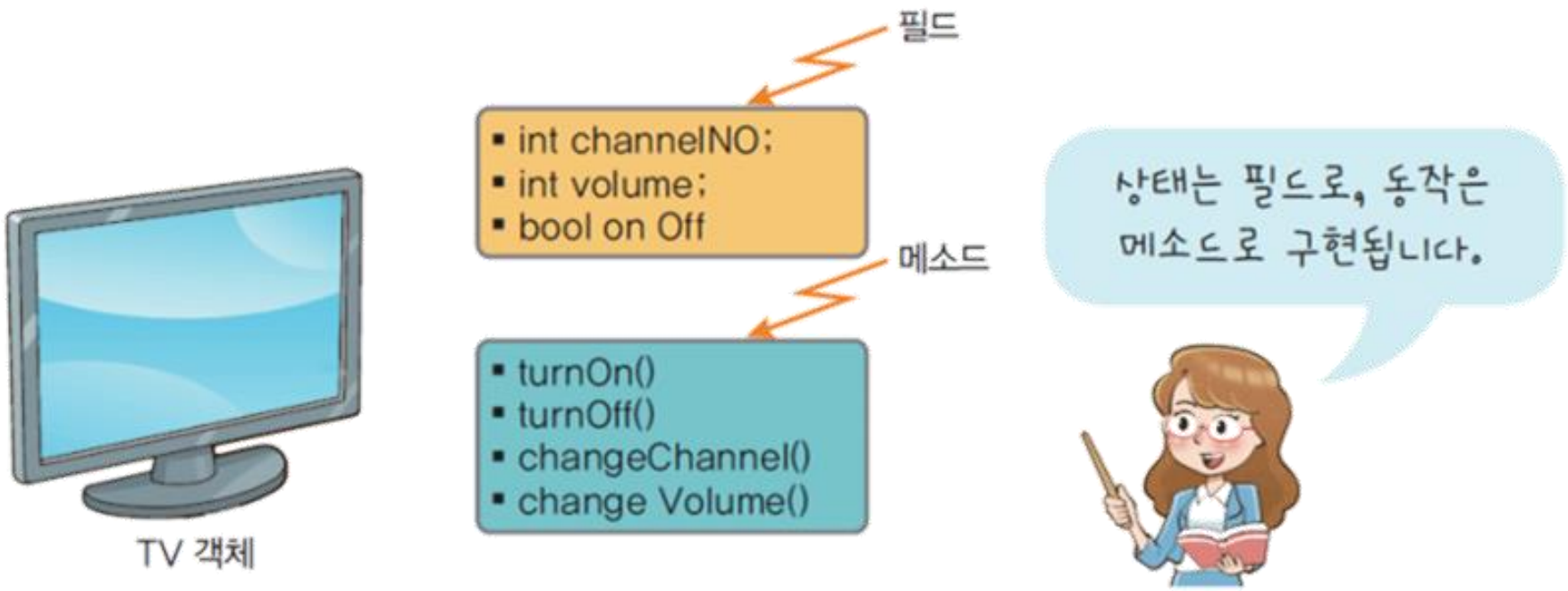
객체 : 상태(변수)와 동작(함수)



들어가기 전에

객체 : 상태(변수)와 동작(함수)

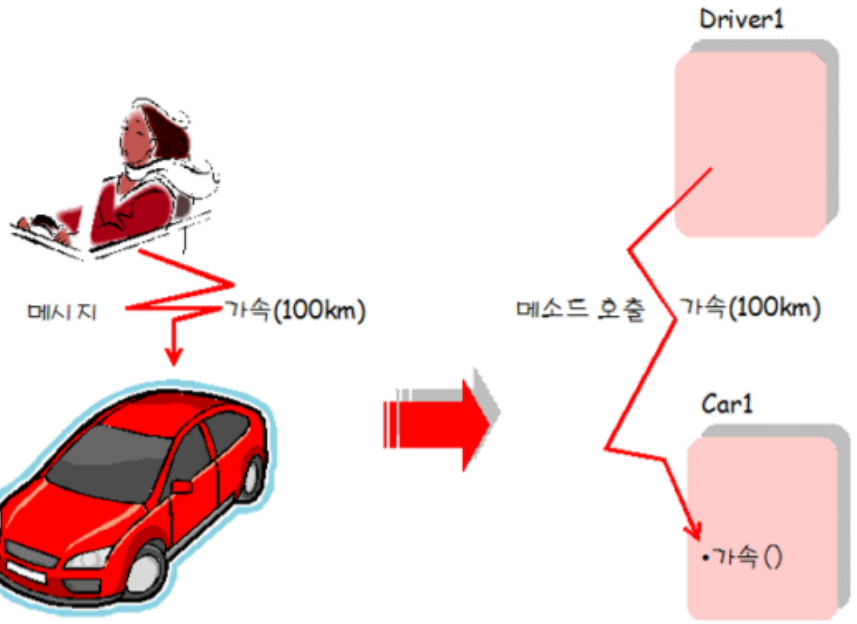
- ✓ 객체(Object)는 상태와 동작을 가지고 있다
- ✓ 객체의 상태 : 객체의 특징 값 ➔ 필드(데이터, 변수)
- ✓ 객체의 동작 : 객체가 취할 수 있는 동작 ➔ 메소드(함수)



들어가기 전에

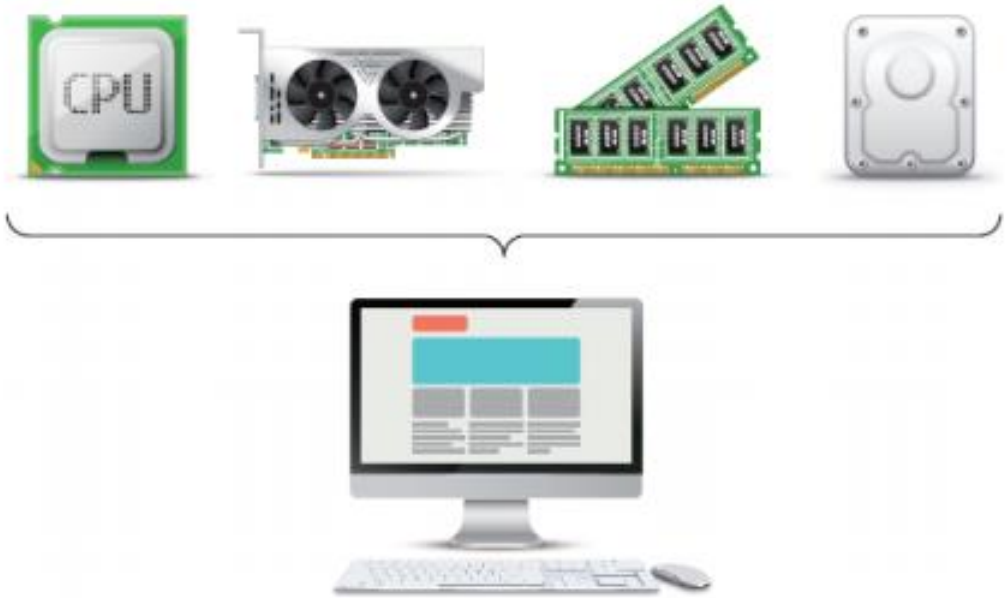
객체와 메시지

- ✓ 객체는 메시지를 통해 다른 객체와 통신하고 서로 상호 작용 한다.
- ✓ 예제
  - ✓ TV 객체의 채널변경 메소드를 리모콘 객체가 호출 : 채널변경(10);
  - ✓ 드라이버 객체에서 자동차 객체의 가속 함수를 호출 : 가속(100km);

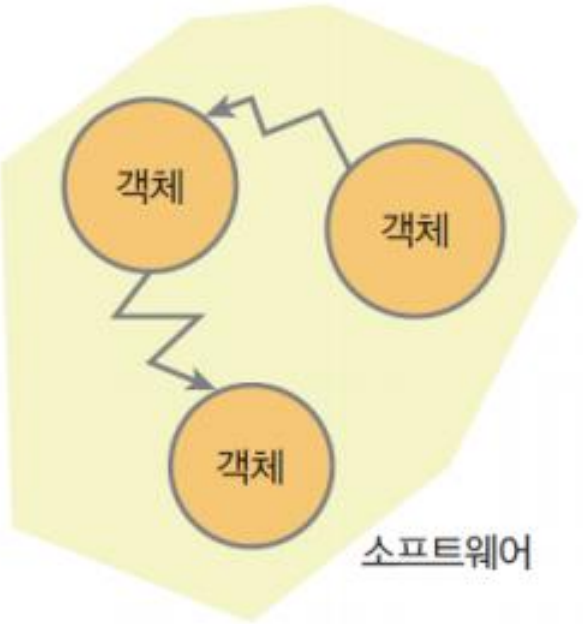


들어가기 전에

객체 지향 프로그래밍



부품을 조립하여 제품을 만들듯이
객체를 조합하여 소프트웨어를 만든다



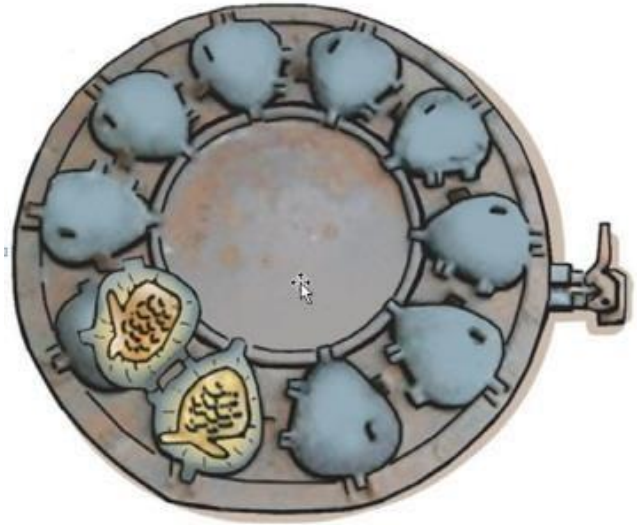
들어가기 전에

C++

C언어 ✚ 클래스
객체지향 특성
가상함수
연산자 중복 정의
다중 상속
템플릿
예외 처리 등 = C++

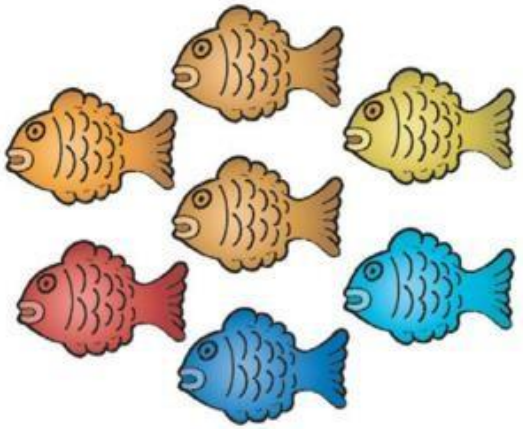
클래스

- ✓ 클래스와 객체
  - ✓ 클래스 : 객체를 만드는 틀
  - ✓ 객체 : 클래스라는 틀에서 생겨난 실체
  - ✓ 객체(object), 실체(instance)는 같은 뜻



클래스
(객체를 정의하는 틀)

객체 생성



객체들 - 실체

클래스의 구조

형식

```
class 클래스이름 {
    자료형 필드1;
    자료형 필드2;
    ....

    반환형 메소드1()      ... }
    반환형 메소드2()      ... }
    ...
}
```

필드 정의
객체의 속성을 나타낸다.

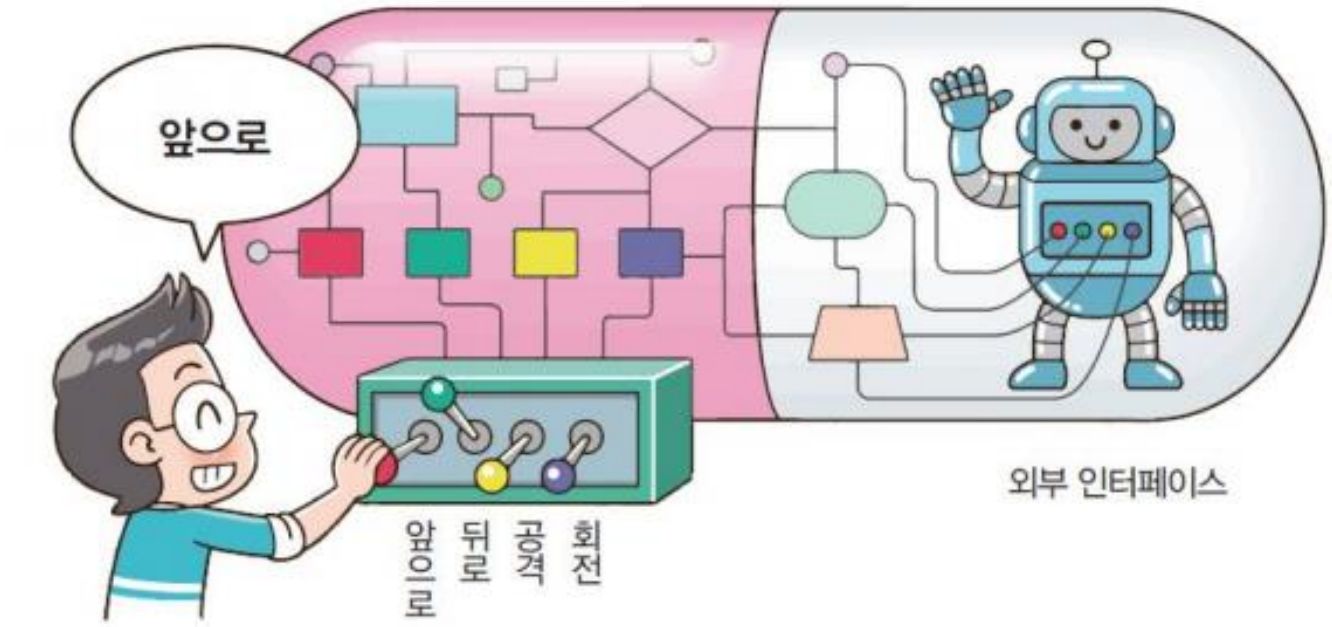
메소드 정의
객체의 동작을 나타낸다.

캡슐화(Encapsulation)

- ✓ 데이터를 캡슐로 싸서 외부의 접근으로부터 보호
- ✓ C++에서 클래스(class 키워드)로 캡슐 표현
- ✓ 장점
  - ✓ 남의 코드(클래스를) 가져다 쓰기 편하다
  - ✓ 남이 코드를(클래스를) 가져다 써도 맘이 편하다

캡슐화와 정보 은닉

- ✓ 정보 은닉(information hiding)은 객체를 캡슐로 싸서 객체의 내부를 보호하는 것이다.
- ✓ 즉 객체의 실제 구현내용을 외부에 감추는 것이다.
- ✓ 객체의 사용 = 공개된 인터페이스(조이스틱)를 통하여 사용 = API



객체 지향 프로그래밍

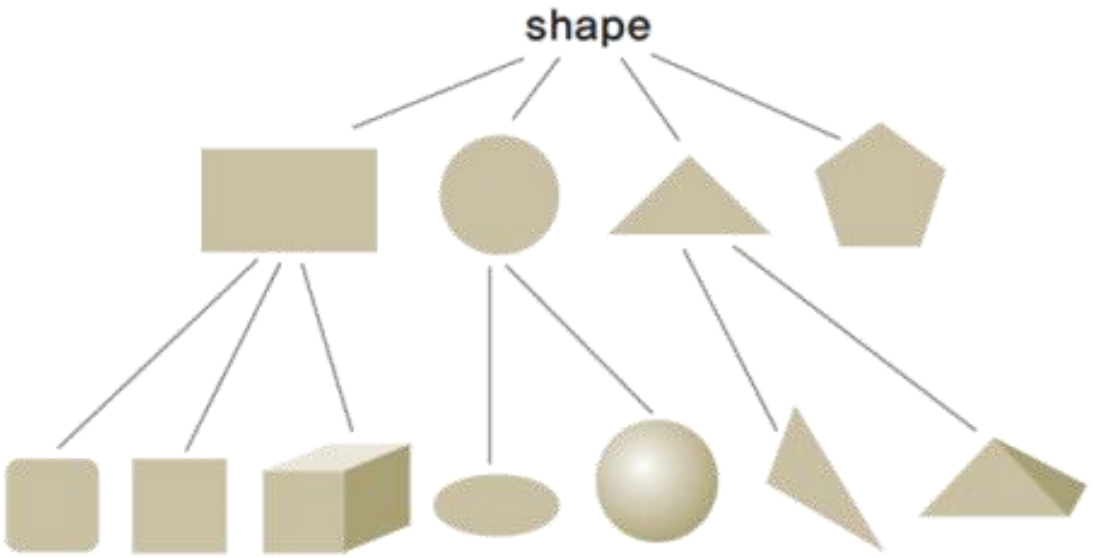
인터페이스가 있을 때

객체 지향 프로그래밍

인터페이스가 없을 때

상속

- ✓ 상속(inheritance): 작성된 클래스(부모 클래스)를 이어 받아서 새로운 클래스(자식 클래스)를 생성하는 기법
- ✓ 기존의 코드를 재활용하기 위한 기법
  - ✓ 공통되는 메소드는 매번 새로 만들어도 되지 않는다
  - ✓ 새로운 메소드를 만들다가 최초의 기획의도에서 벗어날 일이 줄어든다

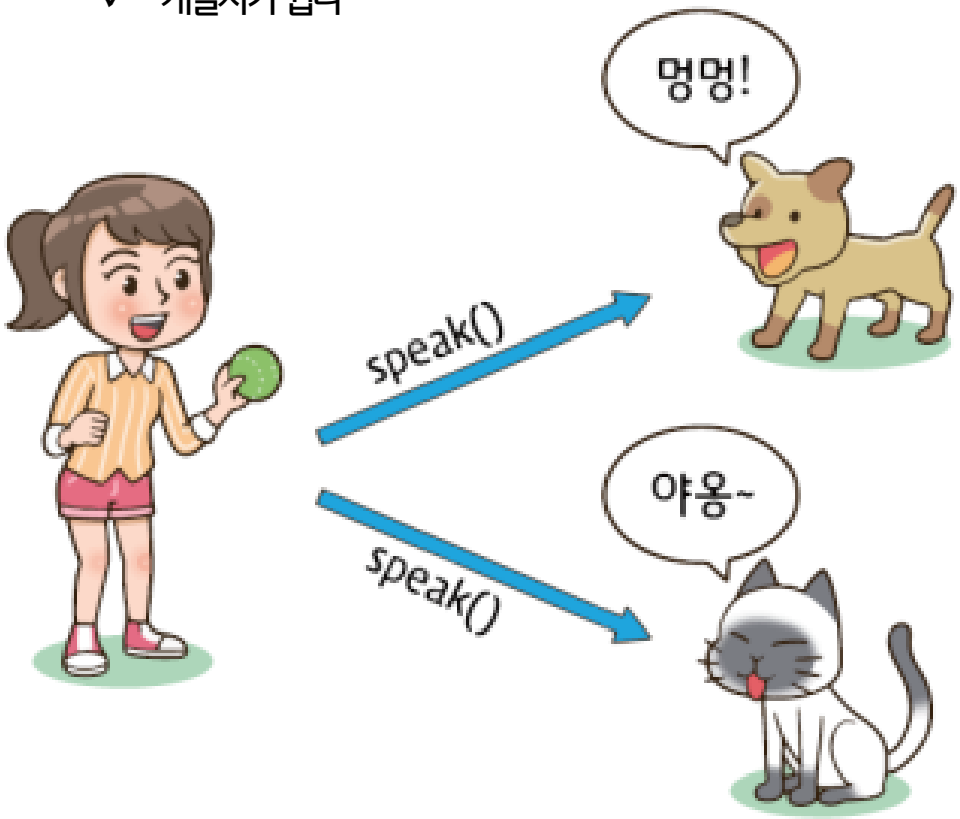


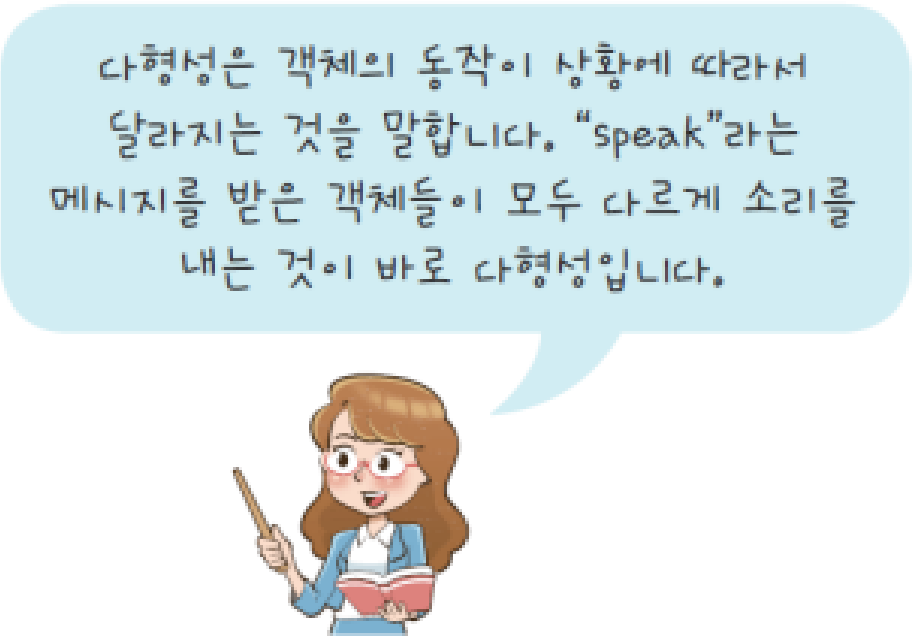
그런데 그것이 실제로 일어났습니다

✓ 인터랙션

객체 지향 특성 : 다형성

- ✓ 하나의 이름(방법)으로 많은 상황에 대처하는 기법
- ✓ 개념적으로 동일한 작업을 하는 멤버 함수들에 똑같은 이름을 부여할 수 있으므로 코드가 더 간단해 진다
  - ✓ 개발자가 쉽다





추상화

✓ 실제 객체를 모델링 할 때 필요한 것만 활용

실제 객체

추상화된 객체

C와 C++

	C	C++
프로그래밍 방식	구조적 프로그래밍	객체지향 프로그래밍
프로그램 분할 방법	기능	객체
구현 단위	함수	클래스
규모	중소형 프로그램 작성에 적합	중대형 프로그램 작성에 적합

Hello World.cpp

```
1   // helloworld.cpp ➔ 주석
2
3   #include <iostream> ➔ 전처리지시자, I/O 스트림 객체
4
5   int main() ➔ main 함수
6   {
7       std::cout << "Hello, World!" << std::endl;
8
9       return 0;
10  }
11
```

C++ 에서의 주석

한 줄 씩 : //

```cpp
#include <iostream>

class CTest {

public:
    CTest() {}  // 생성자
    ~CTest() {} // 소멸자

private: // -> private
    CTest( const CTest& ) {}            // 복사 생성자
    CTest& operator=( const CTest& ) {} // 복사 대입 연산자
};

int main( void )
{
    CTest c1;
    //CTest c2( c1 );    // Error
    //CTest c3 = c1;     // Error

    return 0;
}
```

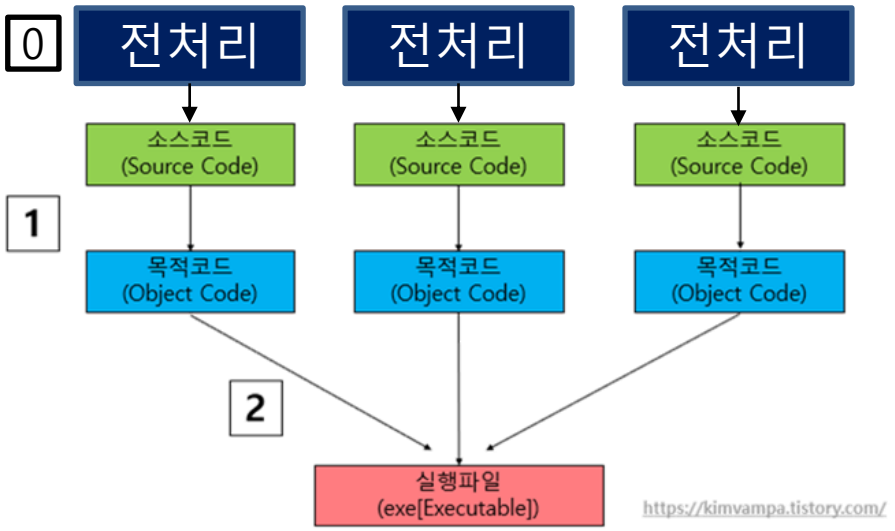
한꺼번에 여러 줄 : /* */

```cpp
#include <iostream>

class CTest {
    /*
public:
    CTest() {}  // 생성자
    ~CTest() {} // 소멸자

    CTest( const CTest& ) {}            // 복사 생성자
    CTest& operator=( const CTest& ) {} // 복사 대입 연산자
    */
};

int main( void )
{
    CTest c1;
    CTest c2( c1 );
    CTest c3 = c1;

    return 0;
}
```

C++ 언어 기초

전처리 지시자

✓ 전처리를 지시하는 애(당연한거 아닌가..)



자주 사용하는 전처리 지시자

#include : 다른 파일의 소스코드를 현재 파일에 삽입

#define : 지시자 이후의 모든 키워드를 지정한 값으로 치환

```
#define TRUE  1
#define FALSE 0
#define PI    3.14159
```

#ifdef #ifndef #endif : 특정 키워드가 define 되어있는지에 따라 지정된 코드 실행

```
#ifndef name
#define name   "wowon"
#endif
```

C++ 언어 기초

main() 함수

✓ 프로그램을 실행할 때 가장 먼저 호출되는 시작점

`int argc` : 매개변수 개수, `char* argv[]` : 매개변수 내용(배열)

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++)
    {
        printf("%s\n", argv[i]);
    }

    return 0;
```

```
c:\project\option\Debug>option.exe Hello C Language
option.exe
Hello
C
Language
```

I/O 스트림

✓ ~~스트림 라이브러리에서 제공하는 입출력 기능~~ ➔ 남이 만든 소스코드

✓ 사용자 콘솔 화면에 데이터를 출력하는 함수 들어있음

✓ 또는 콘솔로 부터 데이터를 입력 받는 함수 들어있음

✓ 자세한건 찾아보세요 : https://modoocode.com/213

Namespace 네임스페이스

- ✓ 코드 내에서 이름이 같은 변수명이나 함수명이 서로 충돌하는 문제를 해결하기 위해 고안 됨
  - ✓ 내가 만든 코드랑 남이만든 코드를 합쳤는데, 우연히 함수 이름이 같은게 있음
  - ✓ 이때 컴파일러는 어떤 함수를 써야하는지 알아낼 방법이 없음

함수 만들 때

```
namespace mycode {
    void foo()
  {
        std::cout << "foo() called in the mycode namespace" << std::endl;
    }
}


void mycode::foo()
{
    std::cout << "foo() called in the mycode namespace" << std::endl;
}
```

함수 불러다 쓸 때

```
using namespace mycode;

int main()
{
    mycode::foo();
    foo();          // mycode::foo(); 와 같다.
    return 0;
}
```

기본 용어들

- ✓ 변수
- ✓ 연산자
- ✓ 타입(자료형)
- ✓ 구조체
- ✓ 조건문
- ✓ 반복문
- ✓ 배열
- ✓ 함수, 메소드

C++ 고급기능

고오급

- ✓ 포인터
- ✓ 동적 메모리
- ✓ 스택과 힙
- ✓ C++의 스트링
  - ✓ C 스타일 스트링 : char 타입 배열 / char* 변수
  - ✓ C++ 언어의 스트링 : string 자료형
  - ✓ 비표준 스트링 : 자신만의 문자열 타입을 만든다
- ✓ 참조형
- ✓ 예외처리