

# 04. 클래스와 객체의 기본

4.1 클래스와 객체의 기본 개념

4.2 string 클래스와 vector 클래스

4.3 C++에서의 클래스 정의 및 사용

## 4.1 클래스와 객체의 기본 개념

# 세상의 모든 것이 객체이다.

## ■ 세상 모든 것이 객체



TV



의자



책



집



카메라

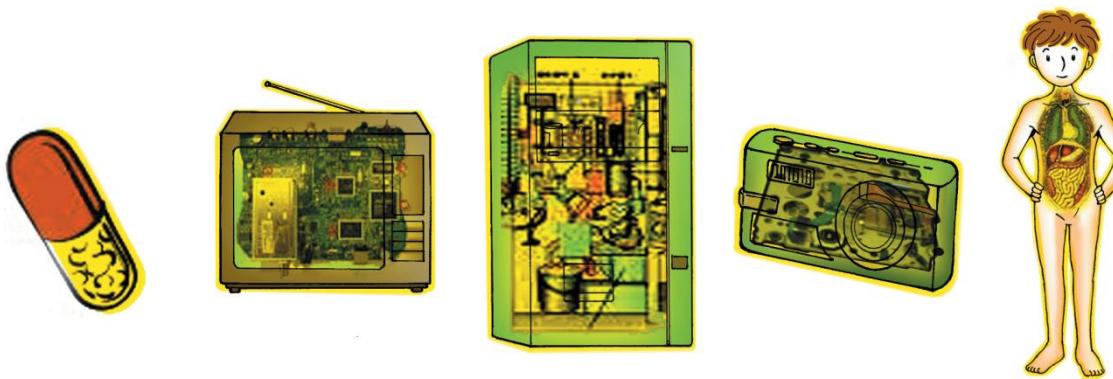


컴퓨터

# 객체는 캡슐화된다.

## ■ 캡슐화(encapsulation)

- 객체의 본질적인 특성
- 객체를 캡슐로 싸서 그 내부를 보호하고 볼 수 없게 함
  - 캡슐에 든 약은 어떤 색인지 어떤 성분인지 보이지 않고, 외부로부터 안전
- 캡슐화 사례

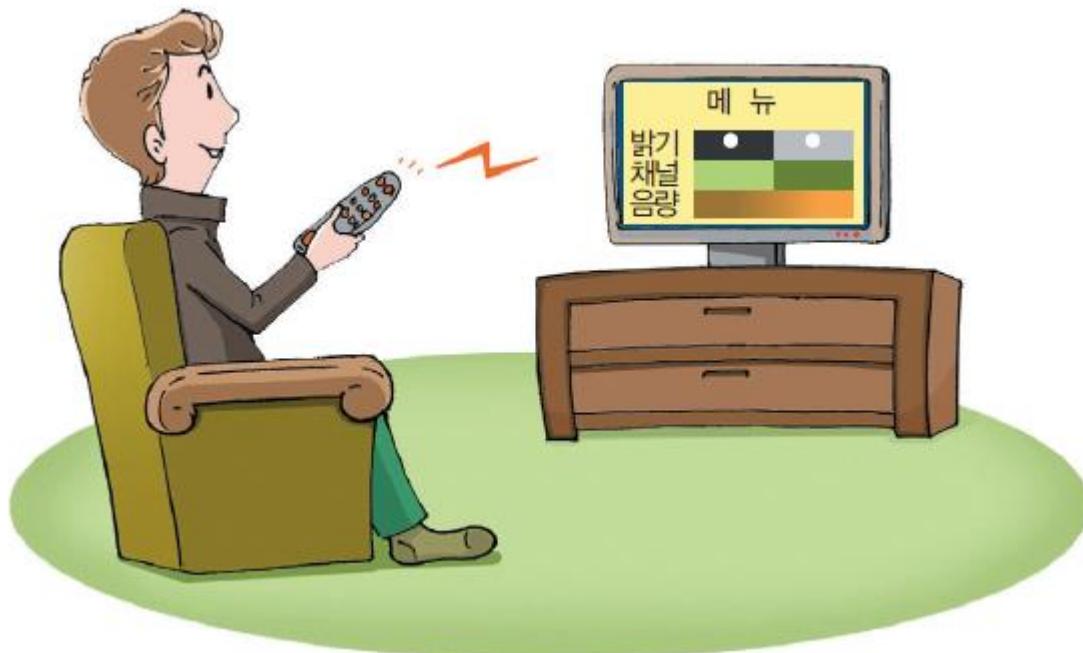


- 캡슐화의 목적
  - 객체 내 데이터에 대한 보안, 보호, 외부 접근 제한

# 객체의 일부 요소는 공개된다.

## ■ 객체의 일부분 공개

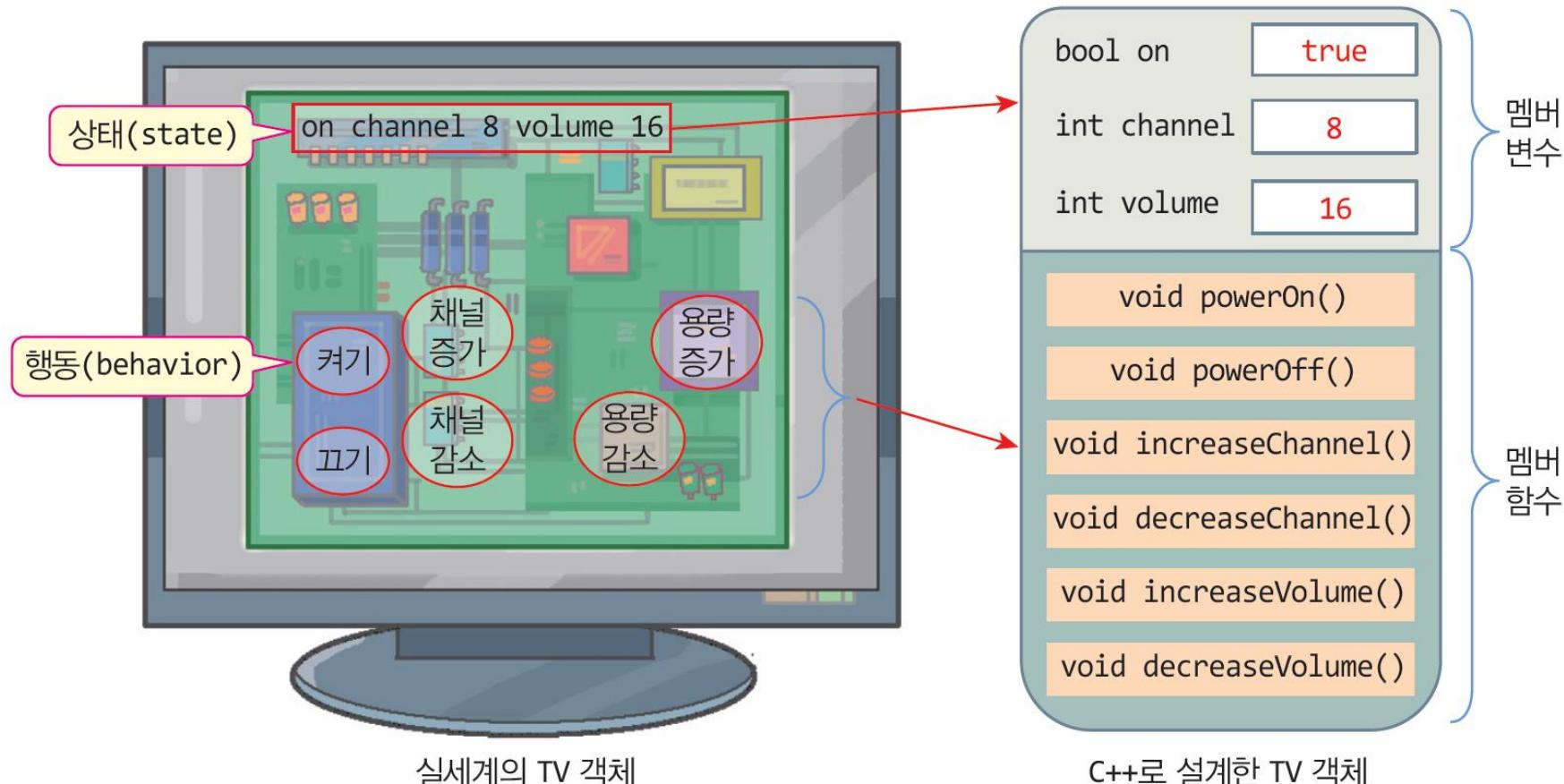
- 외부와의 인터페이스(정보 교환 및 통신)를 위해 객체의 일부분 공개
- TV 객체의 경우, On/Off 버튼, 밝기 조절, 채널 조절, 음량 조절 버튼 노출. 리모콘 객체와 통신하기 위함



# C++ 객체는 멤버 변수와 멤버 함수로 구성된다.

- 객체는 상태(state)와 행동(behavior)으로 구성
- TV 객체 사례
  - 상태
    - on/off 속성 – 현재 작동 중인지 표시
    - 채널(channel) - 현재 방송중인 채널
    - 음량(volume) – 현재 출력되는 소리 크기
  - 행동
    - 켜기(power on)
    - 끄기(power off)
    - 채널 증가(increase channel)
    - 채널 감소(decrease channel)
    - 음량 증가(increase volume)
    - 음량 줄이기(decrease volume)

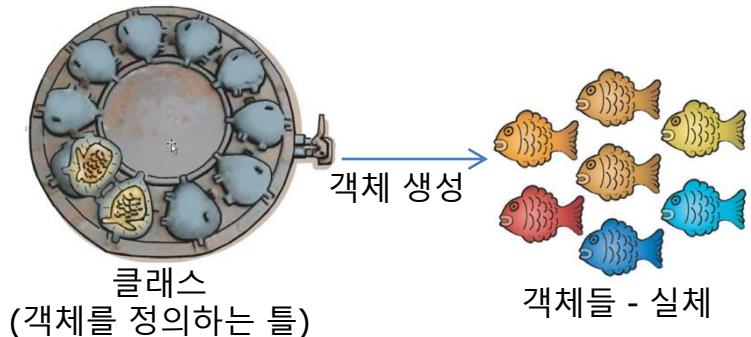
# TV와 C++로 설계된 TV 객체



# C++ 클래스와 C++ 객체

## ■ 클래스(Class)

- 사용자정의 자료형이라 생각하면 편함(일종의 데이터 형 역할)
- 객체를 만들어내기 위해 정의된 설계도, 틀
- 클래스는 객체가 아님. 실체도 아님
- 멤버 변수와 멤버 함수 선언



## ■ 객체(Object)

- 클래스 형에 의해 만들어진 실제 사용되는 변수
- 객체는 생성될 때 클래스의 모양을 그대로 가지고 탄생
- 멤버 변수와 멤버 함수로 구성
- 메모리에 생성, 실체(instance)라고도 부름
- 하나의 클래스 틀에서 찍어낸 여러 개의 객체 생성 가능
- 객체들은 상호 별도의 공간에 생성

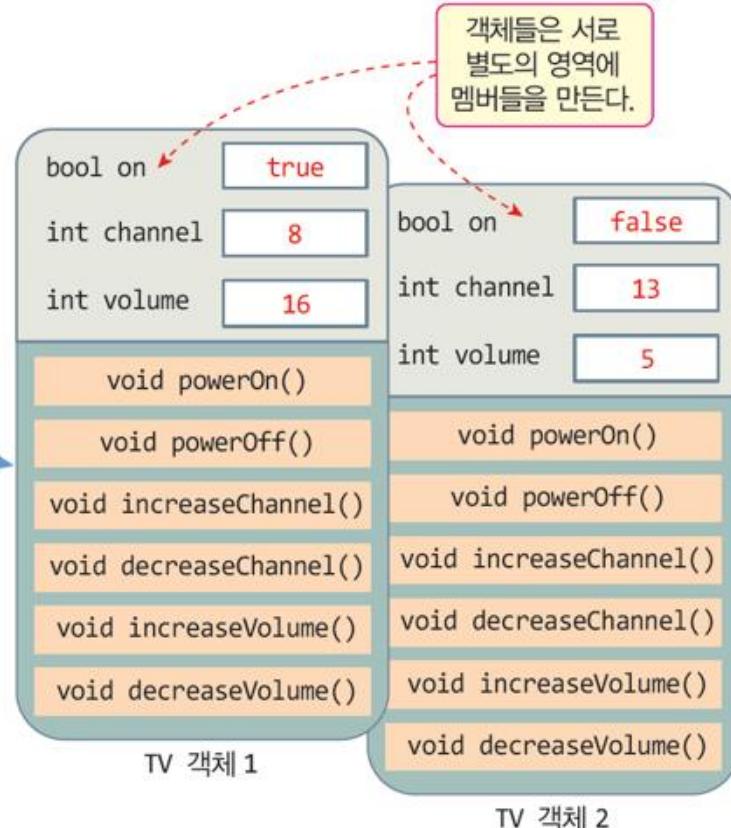
# 클래스와 객체 관계

## C++로 표현한 TV 클래스와 TV 객체들

```
class TV {  
    bool on;  
    int channel;  
    int volume;  
public:  
    void powerOn() { ... }  
    void powerOff() { ... }  
    void increaseChannel() { ... }  
    void decreaseChannel() { ... }  
    void increaseVolume() { ... }  
    void decreaseVolume() { ... }  
};
```

TV 클래스

객체 생성



## 4.2 string 클래스와 vector 클래스

# 이미 만들어진 클래스 사용해보기

---

- **string** 클래스

- 문자열에 대한 처리를 제공

- **vector** 클래스

- 실행 중에 크기를 변경할 수 있는 배열 기능을 제공

# C++에서 문자열을 다루는 **string** 클래스

## ■ C++ 문자열

- C-스트링
- C++ string 클래스의 객체

## ■ string 클래스

- C++ 표준 라이브러리, <string> 헤더 파일에 선언

```
#include <string>
using namespace std;
```

- 가변 크기의 문자열

```
string str = "I love "; // str은 'I', ' ', 'l', 'o', 'v', 'e', ''의 7개 문자로 구성
str.append("C++."); // str은 "I love C++."이 된다. 11개의 문자
```

- 다양한 문자열 연산을 실행하는 연산자와 멤버 함수 포함
  - 문자열 복사, 문자열 비교, 문자열 길이 등
- 문자열, 스트링, 문자열 객체, string 객체 등으로 혼용

# string 객체 생성 및 입출력

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name; // 빈 문자열을 가진 스트링 객체
    string address("서울시 성북구 삼선동 389"); // 문자열 리터럴로 초기화
    string copyAddress(address); // address를 복사한 copyAddress 생성

    // C-스트링(char [] 배열)으로부터 스트링 객체 생성
    char text[] = {'L', 'o', 'v', 'e', ' ', 'C', '+', '+', '\0'};
    string title(text); // "Love C++" 문자열을 가진 title 생성

    cin >> name; // 공백이 입력되면 하나의 문자열로 입력
    cout << str << "\n";
    cout << address << "\n";
    cout << copyAddress << "\n";
    cout << text << "\n";
    cout << title << "\n";

    return 0;
}
```

# 문자열 복사

## ■ string 객체를 이용한 문자열 복사 예

```
#include <string>
using namespace std;

// 중간 생략

string src = "C++ Programming";
string dest;

// 문자열의 내용을 복사한다.
dest = src;

cout << "src = " << src << "\n";
cout << "desc = " << dest << "\n";
```

# 문자열의 길이 계산

- string 객체를 이용한 문자열의 길이 구하는 예

```
#include <string>
using namespace std;

// 중간 생략

string s1;
string s2 = "123";
string s3 = "abcdefg";

cout << "s1 = " << s1.size() << "\n";
cout << "s2 = " << s2.size() << "\n";
cout << "s3 = " << s3.size() << "\n";
```

# 문자열의 결합과 비교(1)

- string 객체를 이용한 문자열의 길이 구하는 예

```
#include <string>
using namespace std;

// 중간 생략

string str1 = "abcde";
string str2 = "fghij";

// 두 문자열을 결합한다.
str1 = str1 + str2;

if ( str1 == "abcdefg hij" )
    cout << "str1 and \"abcdefg hij\" are identical. \n";
if ( "123456" != str1 )
    cout << "\"123456\"" and str1 are NOT identical. \n";
```

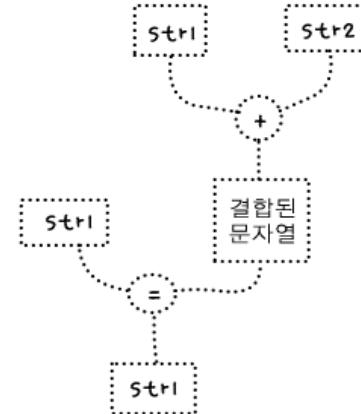
# 문자열의 결합과 비교(2)

## ■ + 연산자를 사용한 문자열 결합

더해진 결과를  
str1에 다시 담는다

str1 = str1 + str2;

str1이 가진 문자열의 뒤에  
str2가 가진 문자열을 더한다



〈기본 타입과 같은 방식으로 계산한다〉

## ■ ==와 != 연산자를 사용한 문자열 비교

if ( str1 (==) "abcdefghijkl" )

str가 가진 문자열과 'abcdefghijkl'의  
내용이 같은지 비교한다

if ( str1 (!=) "abcdefghijkl" )

str가 가진 문자열과 'abcdefghijkl'의  
내용이 다른지 비교한다

# 문자열 검색

## ■ string 객체를 이용한 특정 문자열 검색 예

```
#include <string>
using namespace std;

// 중간 생략

string text = "Napster's pay-to-play service is officially out,"
              "and we have a review of the now-legit Napster."
              "We also size up its companion music player from Samsung.';

// 이 문자열 안에서 'official'이라는 단어의 위치를 찾는다.
cout << "Offset of 'official' = " << text.find("official") << "\n";
```

첫글자 'o'는 전체 문자열에서  
34번째 글자다.  
즉, text[33] == 'o'가 된다

```
String text = "Napster's pay-to-play service is officially out,"
              "and we have a review of the now-legit Napster."
              "We also size up its companion music player from Samsung.';



Offset of 'official' = 34
```

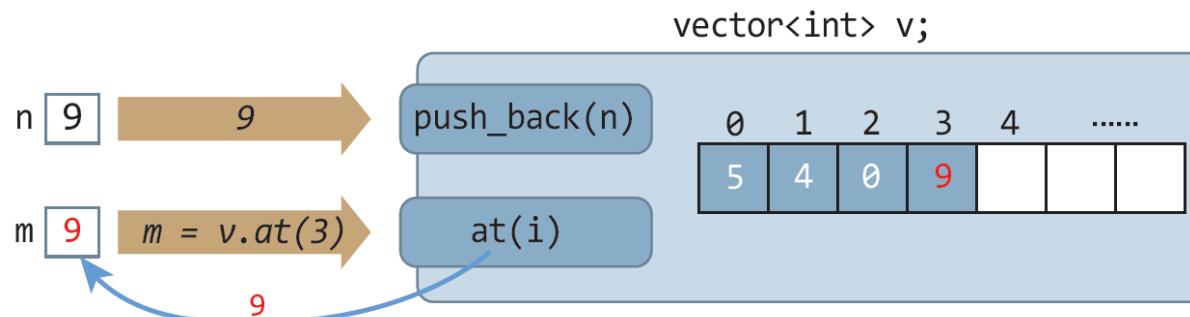
# string 클래스의 주요 멤버 함수들

생성자	특징
string str; string str1("string"); string str2(str1);	디폴트 생성자, 빈문자열을 가지는 str 이름을 가지는 스트링 객체 문자열 상수 "string"값으로 초기화하여 str1 객체 생성 객체 str1 값을 복사한 str2 객체 생성
원소값 접근	특징
str[i] str.at(i) str.substr(position, length)	i번째 인덱스에 위치란 문자값을 입/출력 i번째 인덱스에 위치란 문자값을 입/출력 position 값에서 시작하여 length 값 만큼의 길이를 가지는 문자열 인 subString 값을 리턴함.
할당 / 변경	특징
str1 = str2; str1 += str2; str.empty(); str1 + str2; str.insert(pos, str2); str.remove(pos, length); str1 == str2 str1 != str2 str1 < str2 str1 > str2 str1 <= str2 str1 >= str2 str1.find(str1)	문자저장공간을 설정하고 str2값으로 초기화 str2 문자값을 str1의 뒤에 연결시킴 str이 빈 스트링일 경우 true값을, 아니면 false값을 리턴함 str2 값이 str1의 끝 부분에 연결된 스트링을 리턴함 str2 값을 pos 위치부터 시작하는 str값에 삽입함 pos 위치부터 시작하여 length 크기만큼 substring의 값을 삭제함 값이 서로 동일한지의 여부를 확인하여 boolean 값을 리턴함 스트링의 값을 크기 비교하는 4가지 방법 값의 크기는 사전적 순서에 따라 결정 str1에서 str1값이 처음으로 존재하는 인덱스 값을 리턴함

# vector 클래스

## ■ vector 클래스 : 동적 배열

- 가변 길이 배열을 구현한 제네릭 클래스(Template Class)
  - 개발자가 벡터의 길이에 대한 고민할 필요 없음
  - Template Class이기에 어떤 타입이라도 저장이 가능(Generic Array)
- 원소의 저장, 삭제, 검색 등 다양한 멤버 함수 지원
- 벡터에 저장된 원소는 인덱스로 접근 가능 (인덱스는 0부터 시작)
- 원소들이 연속적인 메모리 공간에 할당되어 메모리가 적게 사용되고, 임의의 위치로의 접근이 좋음 : operator []로 접근이 가능
- 연속된 메모리 공간에 할당되어 있다는 점에서 끝에 삽입/삭제가 일어날 경우 속도가 빠름.



# vector 클래스의 주요 멤버와 연산자

멤버와 연산자 함수	설명
<code>push_back(element)</code>	벡터의 마지막에 element 추가
<code>at(int index)</code>	index 위치의 원소에 대한 참조 리턴
<code>begin()</code>	벡터의 첫 번째 원소에 대한 참조 리턴
<code>end()</code>	벡터의 끝(마지막 원소 다음)을 가리키는 참조 리턴
<code>empty()</code>	벡터가 비어 있으면 true 리턴
<code>erase(iterator it)</code>	벡터에서 it가 가리키는 원소 삭제. 삭제 후 자동으로 벡터 조절
<code>insert(iterator it, element)</code>	벡터 내 it 위치에 element 삽입
<code>size()</code>	벡터에 들어 있는 원소의 개수 리턴
<code>operator[]( )</code>	지정된 원소에 대한 참조 리턴
<code>operator=( )</code>	이 벡터를 다른 벡터에 치환(복사)

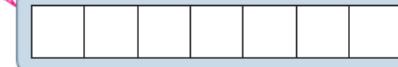
# vector 다루기 사례

vector 생성

```
vector<int> v;
```

정수 벡터  
생성

```
vector<int> v
```

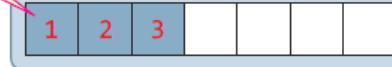


정수 원소 삽입

```
v.push_back(1);  
v.push_back(2);  
v.push_back(3);
```

정수 삽입

v



원소 개수 s  
벡터의 용량 c

```
int s = v.size(); // s는 3  
int c = v.capacity(); // c는 7
```

s = 3  
c = 7

원소 값 접근

```
v.at(2) = 5;  
int n = v.at(1);
```

n = 2

v

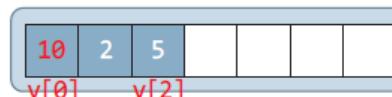


원소 값 접근

```
v[0] = 10;  
int m = v[2]; // m은 5
```

m = 5

v



# vector 클래스 사용하기

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v; // 정수만 삽입 가능한 벡터 생성

    v.push_back(1); // 벡터에 정수 1 삽입
    v.push_back(2); // 벡터에 정수 2 삽입
    v.push_back(3); // 벡터에 정수 3 삽입

    for(int i=0; i<v.size(); i++) // 벡터의 모든 원소 출력
        cout << v[i] << " "; // v[i]는 벡터의 i 번째 원소
    cout << endl;

    v[0] = 10; // 벡터의 첫 번째 원소를 10으로 변경
    int n = v[2]; // n에 3이 저장
    v.at(2) = 5; // 벡터의 3 번째 원소를 5로 변경

    for(int i=0; i<v.size(); i++) // 벡터의 모든 원소 출력
        cout << v[i] << " "; // v[i]는 벡터의 i 번째 원소
    cout << endl;
}
```

1 2 3  
10 2 5

## 4.3 C++에서의 클래스 및 객체 정의 및 사용

# C++ 클래스 만들기

- **class 키워드**를 사용하여 클래스 정의
- 클래스는 **멤버 변수**와 **멤버 함수**를 갖는다.
  - 멤버 변수
    - 클래스의 멤버인 변수
    - 멤버 변수의 이름을 정할 때는 m\_ 또는 \_를 변수 이름 앞에 접두사로 사용
  - 멤버함수
    - 클래스의 멤버인 함수
    - 멤버 함수는 해당 클래스의 모든 멤버를 직접 사용 가능  
(따로 객체 이름을 지정할 필요가 없음)
- 접근 지정자 : **private, protected, public** 키워드
  - 클래스의 멤버를 정의할 때 **접근 지정자**를 사용
  - 접근 지정자를 사용해서 정보 은닉과 캡슐화 구현

# C++ 클래스 만들기

## ■ 클래스 작성

- 멤버 변수와 멤버 함수로 구성
- 클래스 선언부와 클래스 구현부로 구성

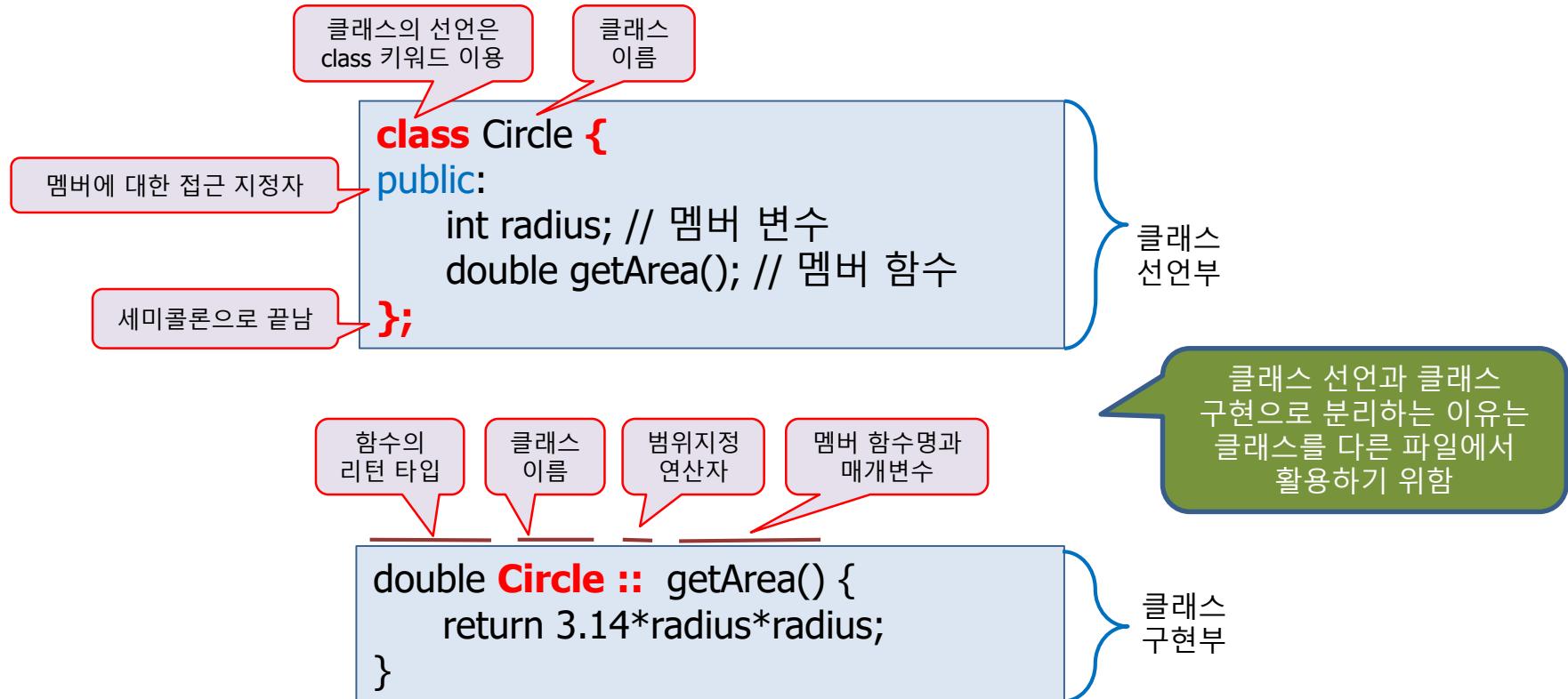
## ■ 클래스 선언부(**class declaration**)

- **class** 키워드를 이용하여 클래스 선언
- **멤버 변수와 멤버 함수** 선언
  - 멤버 변수는 클래스 선언 내에서 초기화할 수 없음
  - 멤버 함수는 원형(prototype) 형태로 선언
- 멤버에 대한 **접근 권한** 지정
  - private, public, protected 중의 하나
  - 디폴트는 private
  - public : 다른 모든 클래스나 객체에서 멤버의 접근이 가능함을 표시

## ■ 클래스 구현부(**class implementation**)

- 클래스에 정의된 모든 멤버 함수 구현. 클래스 선언부에 함께 구현할 수도 있음.

# Circle 클래스 사례



# 멤버 함수의 클래스 선언부 구현

## ■ 클래스 선언부에 구현된 멤버 함수 : 자동 인라인 함수

- 컴파일러에 의해 자동으로 인라인 처리
- inline으로 선언할 필요 없음.
- 모든 함수가 자동 인라인 함수 가능

```
class Circle {  
private:  
    int radius;  
public:  
    double getArea();  
};  
  
inline double Circle::getArea() {  
    return 3.14*radius*radius;  
}
```

inline  
멤버  
함수

(a) 멤버함수를 inline으로 선언하는 경우

```
class Circle {  
private:  
    int radius;  
public:  
    double getArea() { // 자동 인라인 함수  
        return 3.14*radius*radius;  
    }  
};
```

(b) 자동 인라인 함수로 처리되는 경우

# Circle 클래스의 객체 생성 및 활용

```
#include <iostream>
using namespace std;

class Circle {
public:
    int radius;
    double getArea();
};

double Circle::getArea() {
    return 3.14*radius*radius;
}

int main() {
    Circle donut;
    donut.radius = 1; // donut 객체의 반지름을 1로 설정
    double area = donut.getArea(); // donut 객체의 면적 알아내기
    cout << "donut 면적은 " << area << endl;

    Circle pizza;
    pizza.radius = 30; // pizza 객체의 반지름을 30으로 설정
    area = pizza.getArea(); // pizza 객체의 면적 알아내기
    cout << "pizza 면적은 " << area << endl;
}
```

Circle 선언부

Circle 구현부

객체 donut 생성

donut의 멤버 변수 접근

donut의 멤버 함수 호출

donut 면적은 3.14  
pizza 면적은 2826

# 객체 생성 및 활용

## ■ 객체 이름 및 객체 생성

```
Circle donut; // 이름이 donut 인 Circle 타입의 객체 생성
```

객체의 타입.  
클래스 이름

객체 이름

## ■ 객체의 멤버 변수 접근

```
donut.radius = 1; // donut 객체의 radius 멤버 값을 1로 설정
```

객체 이름

멤버 변수

객체 이름과  
멤버 사이에  
. 연산자

## ■ 객체의 멤버 함수 접근

```
double area = donut.getArea(); //donut 객체의 면적 알아내기
```

객체 이름

멤버 함수 호출

객체 이름과  
멤버 사이에  
. 연산자

# 객체 이름과 생성, 접근 과정

(1) Circle donut;

객체 이름

객체가 생성되면  
메모리가 할당된다.

int radius

double getArea() {...}

donut 객체

(2) donut.radius = 1;

int radius

1

double getArea() {...}

donut 객체

(3) double area = donut.getArea();

main()

area

3.14

int radius

1

double getArea() {...}

donut 객체

# 객체의 생성 및 사용

## ■ 멤버 변수의 메모리 할당 시점

- 클래스를 정의한다고 해서 멤버 변수가 메모리에 할당되지 않음.
- **클래스의 객체를 생성하는 시점에 멤버 변수가 메모리에 할당됨.**

## ■ 멤버 변수나 멤버 함수에 접근 방법

- 일반 객체 : . 연산자 사용
- 동적 객체 : → 연산자 사용

```
Circle c1;      // 일반 객체  
c1.radius = 2;  
Circle *c2 = new Circle; // 동적 객체, 객체에 대한 포인터  
p →radius = 3;
```

## ■ 멤버 함수

- 객체마다 만들어지는 것이 아니라 클래스 전체에 대해서 한번만 정의하고 같은 클래스의 객체들이 공유해서 사용

## ■ 멤버함수에서 사용되는 멤버 변수

- 멤버 함수를 호출한 객체의 것

# 멤버 함수

- 멤버 함수도 함수이므로 **오버로딩**하거나 **디폴트 인자** 지정 가능
- 멤버 함수와 전역 함수의 차이점
  - 멤버 함수를 호출하려면 객체 이름을 지정해야 한다.

```
Circle c1;  
double area = c1.getArea(); // Circle 클래스의 멤버 함수 호출  
getArea(); // 전역 함수를 의미하므로 컴파일 에러
```

- 멤버 함수 안에서 같은 클래스에 정의된 멤버 변수나 다른 멤버 함수에 접근할 때는 객체 이름을 지정하지 않는다.

```
double Circle::getArea() {  
    return 3.14 * radius * radius; // 객체 이름을 지정하지 않음  
}
```

- 멤버 함수 안에서는 접근 자정자에 상관없이 클래스의 모든 멤버를 사용 가능함.

# 다음 수업

## ■ 생성자와 소멸자

- 1\_ 생성자와 소멸자
- 2\_ 분할컴파일