Ch09 클래스와 객체

객체 지향 vs 절차 지향

특징	절차지향	객체지향
구조	함수 중심, 데이터와 로직 분리	객체 중심, 데이터와 로직 통합
유지보수	코드 복잡성이 증가하면 어려움	상대적으로 용이
재사용성	낮음	높음
데이터 보호	데이터 보호 어려움	캡슐화를 통해 데이터 보호 가능

#include <iostream>과 std

- std
  - C++ 표준에서 정의한 **이름 공간(namespace)** 중 하나
    - <iostream> 헤더 파일에 선언된 모든 이름: std 이름 공간 안에 있음
    - cout, cin, endl 등
- <iostream>이 통째로 std 이름 공간 내에 선언
  - <iostream> 헤더 파일을 사용하려면 다음 코드 필요

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

#include <iostream>

- <iostream> 헤더 파일
  - 표준 입출력을 위한 클래스와 객체, 변수 등이 선언됨
    - ios, istream, ostream, iostream 클래스 선언
    - cout, cin, <<, >> 등 연산자 선언

이름 충돌 사례

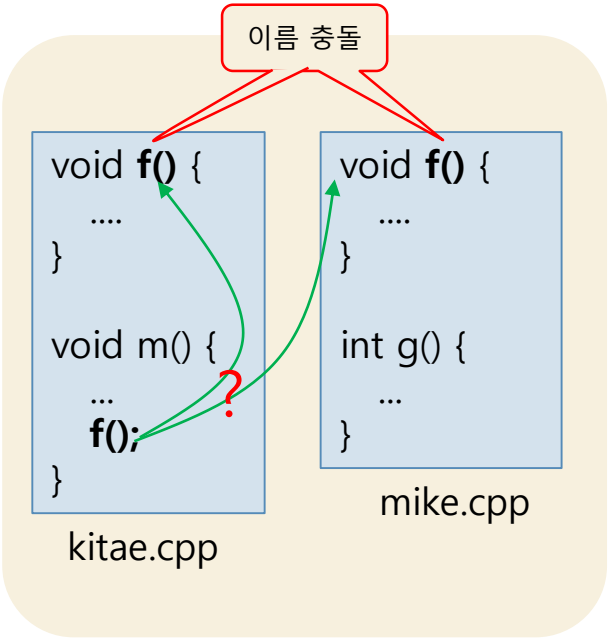
우리 아파트에 여러 명의 마이클이 산다.
마이클을 부를 때, 1동::마이클, 2동::마이클로 부른다.

namespace 개념

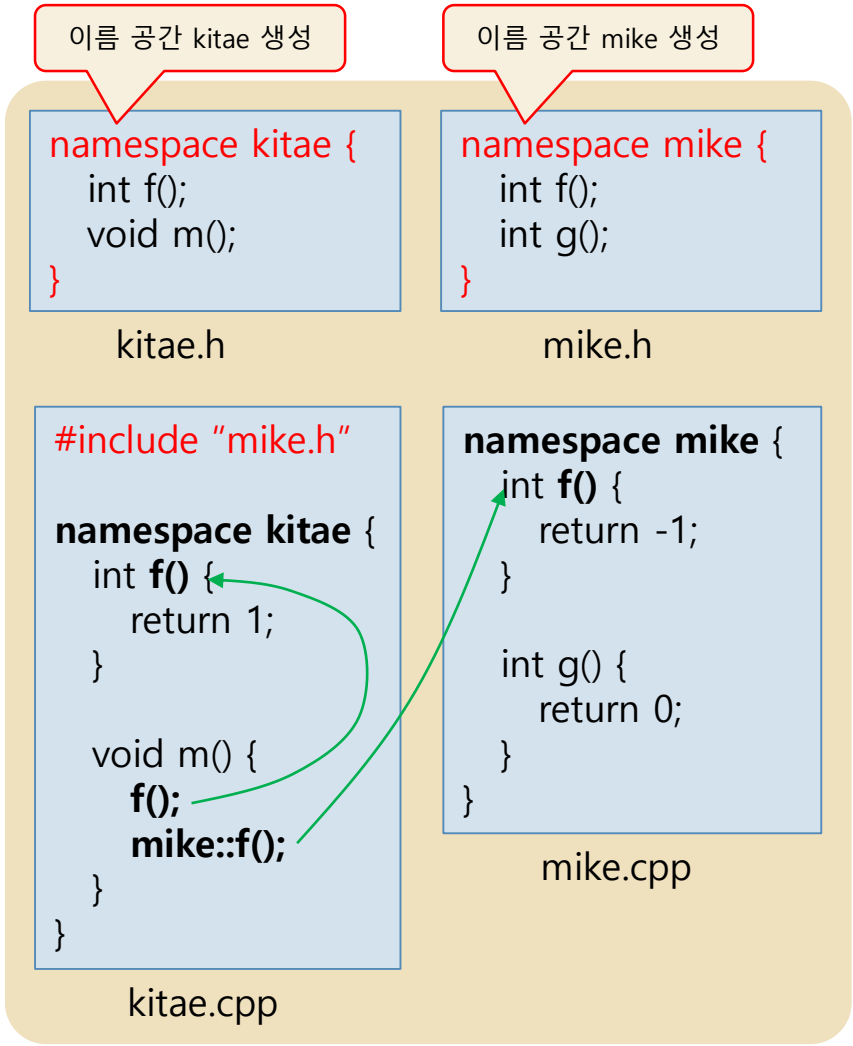
- 이름(identifier) 충돌이 발생하는 경우
  - 여러 명이 서로 나누어 프로젝트를 개발하는 경우
  - 오픈 소스 혹은 다른 사람이 작성한 소스나 목적 파일을 가져와서 컴파일하거나 링크하는 경우
  - 해결하는데 많은 시간과 노력이 필요
- namespace 키워드
  - 이름 충돌 해결
    - 2003년 새로운 C++ 표준에서 도입
  - 개발자가 자신만의 이름 공간을 생성할 수 있도록 함
    - 이름 공간 안에 선언된 이름은 다른 이름공간과 별도 구분
- 이름 공간 생성 및 사용(자세한 것은 부록 B 참고)

```
namespace kitae { // kitae 라는 이름 공간 생성
   ..... // 이 곳에 선언된 모든 이름은 kitae 이름 공간에 생성된 이름
}
```

  - 이름 공간 사용
    - 이름 공간 :: 이름



(a) kitae와 mike에 의해 작성된 소스를 합치면 f() 함수의 이름 충돌. 컴파일 오류 발생



(b) 이름 공간을 사용하여 f() 함수 이름의 충돌 문제 해결

#include <iostream>과 std

- <iostream>이 통째로 std 이름 공간 내에 선언
  - <iostream> 헤더 파일을 사용하려면 다음 코드 필요

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

printf()는 잊어라!

C 언어에서 사용했던 printf를 더 이상 C++에서 사용하지 말기 바란다. printf()나 scanf() 등을 사용하여 구석기 시대로 회귀한다면, 더 이상 C++ 프로그래머로서의 미래는 없다.

cout과 << 연산자를 이용한 출력

- cout
  - cout은 C++ 표준 출력 스트림으로, 데이터를 출력하는 데 사용
- << 연산자
  - 데이터를 출력 스트림(예: std::cout)에 삽입하여 출력하는 데 사용

```
cout << "hello₩n";
```

- Endl
  - 줄바꿈과 출력 버퍼 플러시를 수행.

C++ 프로그램에서 출력하기

ch09_ex1.cpp

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

int main() {
   cout << "hello₩n";
   cout << "hi " << endl;
   cout << "end " << endl;

   return 0;
}
```

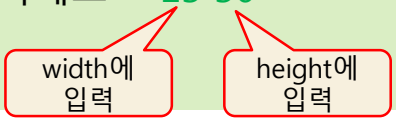
```
hello
hi
end
```

cin과 >> 연산자를 이용한 키 입력

- cin
  - 표준 입력 장치인 키보드를 연결하는 C++ 입력 스트림 객체
- >> 연산자
  - 스트림 추출 연산자(stream extraction operator)
    - C++ 산술 시프트 연산자(>>)가 <iostream> 헤더 파일에 스트림 추출 연산자로 재정의됨
    - 입력 스트림에서 값을 읽어 변수에 저장
  - 연속된 >> 연산자를 사용하여 여러 값 입력 가능

```
cout << "너비와 높이를 입력하세요>>";
cin >> width >> height;
cout << width << '₩n' << height << '₩n';
```

```
너비와 높이를 입력하세요>>23 36
23
36
```



C++ 프로그램에서 키 입력 받기

ch09_ex2.cpp

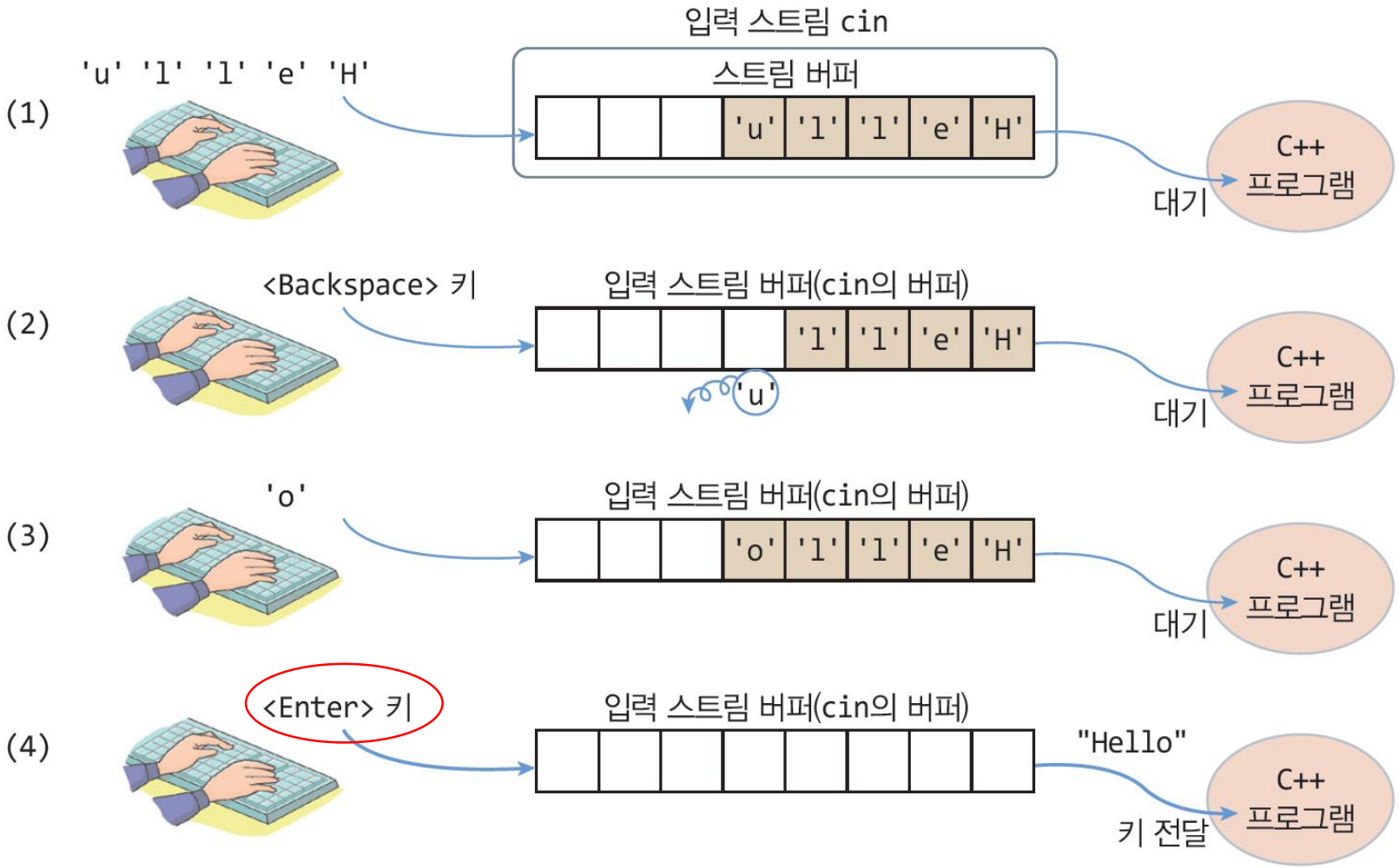
```
#include <iostream>
using namespace std;

int main() {
   cout << "너비를 입력하세요>>";

   int width;
   cin >> width; // 키보드로부터 너비를 읽어 width 변수에 저장

   cout << "높이를 입력하세요>>";

   int height;
   cin >> height; // 키보드로부터 높이를 읽어 height 변수에 저장

   int area = width*height;  // 사각형의 면적 계산
   cout << "면적은 " << area << "₩n"; // 면적을 출력하고 다음 줄로 넘어감
}
```

```
너비를 입력하세요>>3
높이를 입력하세요>>5
면적은 15
```

<Enter> 키를 칠 때 변수에 값 전달

- cin의 특징
  - 입력 버퍼를 내장하고 있음
  - <Enter>키가 입력될 때까지 입력된 키를 입력 버퍼에 저장
    - 도중에 <Backspace> 키를 입력하면 입력된 키 삭제
- >> 연산자
  - <Enter>키가 입력되면 비로소 cin의 입력 버퍼에서 키 값을 읽어 변수에 전달

cin으로부터 키 입력 받는 과정(11.1절)



C++ 문자열

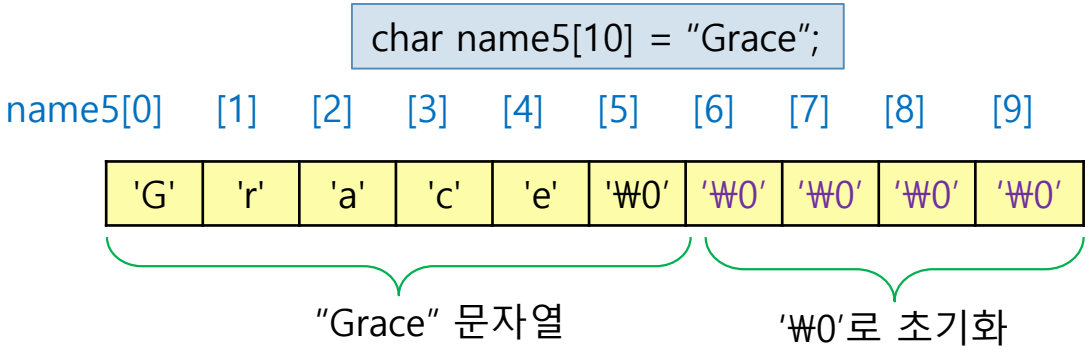
- C++의 문자열 표현 방식 : 2가지
  - **C-스트링 방식** – '₩0'로 끝나는 문자 배열

C-스트링 문자열

```
char name1[6] = {'G', 'r', 'a', 'c', 'e', '₩0'}; // name1은 문자열 "Grace"
```

단순 문자 배열

```
char name2[5] = {'G', 'r', 'a', 'c', 'e'}; // name2는 문자열이 아니고 단순 문자 배열
```



  - **string 클래스 이용**
    - <string> 헤더 파일에 선언됨
    - 다양한 멤버 함수 제공, 문자열 비교, 복사, 수정 등

C-스트링 방식으로 문자열 다루기

- C-스트링으로 문자열 다루기
  - C 언어에서 사용한 함수 사용 가능
    - strcmp(), strlen(), strcpy() 등
  - <cstring>이나 <string.h> 헤더 파일 include

```
#include <cstring> 또는
#include <string.h>
...
int n = strlen("hello");
```

  - <cstring> 헤더 파일을 사용하는 것이 바람직함
    - <cstring>이 C++ 표준 방식

표준 C++ 헤더 파일은 확장자가 없다

- 표준 C++에서 헤더 파일 확장자 없고, std 이름 공간 적시
  - #include <iostream>
  - using namespace std;

- 헤더 파일의 확장자 비교

언어	헤더 파일 확장자	사례	설명
C	.h	〈string.h〉	C/C++ 프로그램에서 사용 가능
C++	확장자 없음	〈cstring〉	using namespace std;와 함께 사용해야 함

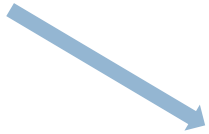
cin을 이용한 문자열 입력

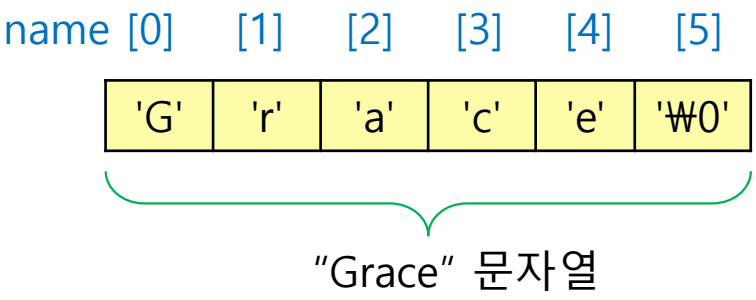
- 문자열 입력

```
char name[6]; // 5 개의 문자를 저장할 수 있는 char 배열
cin >> name; // 키보드로부터 문자열을 읽어 name 배열에 저장한다.
```

Grace

키 입력





키보드에서 문자열 입력 받고 출력

ch09_ex3.cpp

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

int main() {
   cout << "input name : ";

   char name[11]; // 한글은 5개 글자, 영문은 10까지 저장
   cin >> name;
   cout << "My name is " << name ; // 이름을 출력한다.
}
```

이름을 입력하세요>>mary
My name is mary

빈 칸 없이 키 입력해야 함

이름을 입력하세요>>m ary
My name is m

빈 칸을 만나면 문자 열 입력 종료

C-스트링을 이용하여 암호가 입력되면 프로그램을 종료하는 예


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

int main() {
   char password[11];
   cout << "-- Login -- ₩n";
   while(true) {
      cout << "password : ";
      cin >> password;
      if(strcmp(password, "C++") == 0) {
         cout << "Success₩n";
         break;
      }
      else
         cout << "Fail₩n" << endl;
   }
}
```

strcmp() 함수를 사용하기 위한 헤더 파일

```
-- Login --
password : C++
Success
```

빈 칸 없이 키 입력해야 함

```
-- Login --
password : C
Fail
```

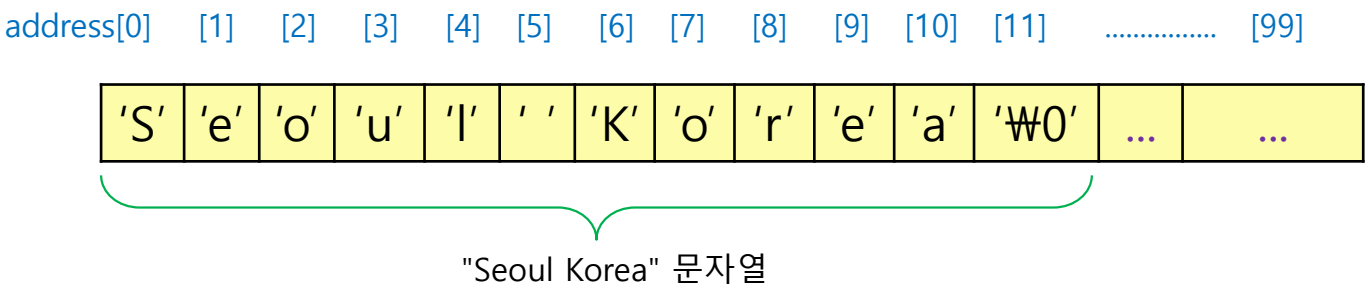
cin.getline()으로 공백이 낀 문자열 입력

- **공백이 낀 문자열**을 입력 받는 방법
- cin.getline(char buf[], int size, char delimitChar)
  - buf에 최대 size-1개의 문자 입력. 끝에 '₩0' 붙임
  - delimitChar를 만나면 입력 중단. 끝에 '₩0' 붙임
    - delimitChar의 디폴트 값은 '₩n'(<Enter>키)

```
char address[100];
cin.getline(address, 100, '₩n');
```

최대 99개의 문자를 읽어 address 배열에 저장. 도중에 <Enter> 키를 만나면 입력 중단

사용자가 'Seoul Korea<Enter>'를 입력할 때,



예제 2–6 cin.getline()을 이용한 문자열 입력


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

int main() {
   cout << "Input Address : ";

   char address[100];
   cin.getline(address, 100, '₩n'); // 키보드로부터 주소 읽기

   cout << "My address is " << address ; // 주소 출력
}
```

```
Input Address : 강원도 춘천시 효자1동
My address is 강원도 춘천시 효자1동
```

빈칸이 있어도 <Enter> 키가 입력 될 때까지 하나의 문자열로 인식

헤더 파일에는 무엇이 들어 있는가?

- 질문1) <cstring>파일에 strcpy() 함수의 코드가 들어 있을까?
  - (1) strcpy() 함수의 코드가 들어 있다(O, X).
    - 답은 X
  - (2) strcpy() 함수의 원형이 선언되어 있다(O, X).
    - 답은 O

- 질문 2) 그러면 strcpy() 함수의 코드는 어디에 있는가?
  - 답) strcpy() 함수의 코드는 컴파일된 바이너리 코드로, 비주얼 스튜디오가 설치된 lib 폴더에 libcmt.lib 파일에 들어 있고
  - 링크 시에 strcpy() 함수의 코드가 exe에 들어간다.

- 질문 2) 그러면 헤더 파일은 왜 사용되는가?
  - 답) 사용자 프로그램에서 strcpy() 함수를 호출하는 구문이 정확한지 확인하기 위해 컴파일러에 의해 필요

C++ 입출력 연습문제

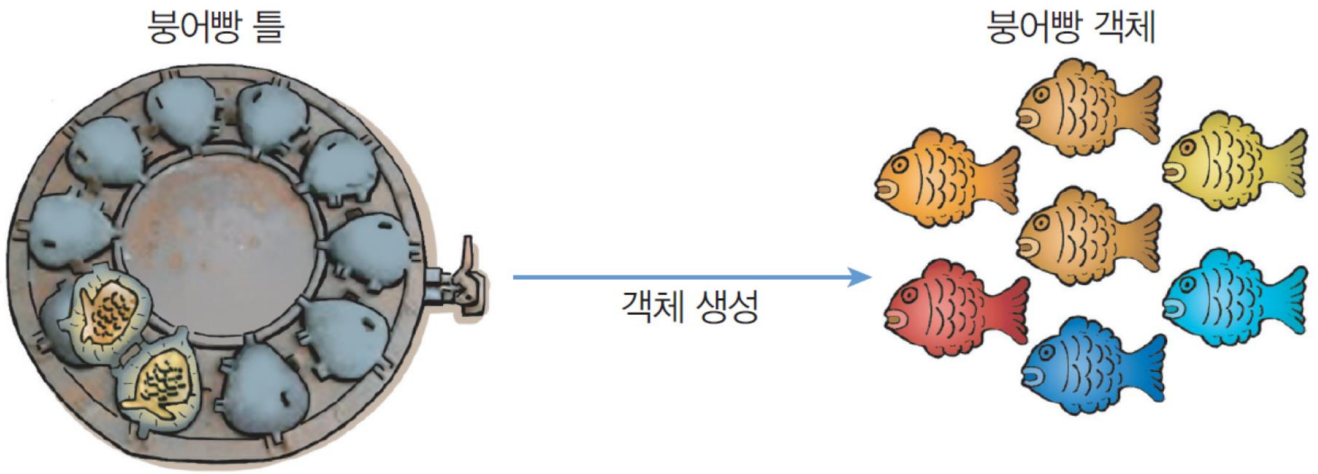
- 아래 내용을 출력하세요
  - 변수 int x=5; int y=1;
  - 변수 x+y의 값을 출력
- 키보드로 2개의 정수를 입력 받아서 두개의 합, 곱의 값을 출력하세요

입력 예)	출력 예)
1 3	4 3

- 키보드로 부터 내 이름과 학번을 입력 받아 출력하세요
  - 입력 예) 203023 김철수

클래스와 객체

- 클래스 : 객체를 생성하기 위한 틀 또는 설계도
- 객체 : 클래스를 기반으로 생성된 실체



(a) 붕어빵 틀과 붕어빵 객체들

C++ 객체는 멤버 함수와 멤버 변수로 구성된다.

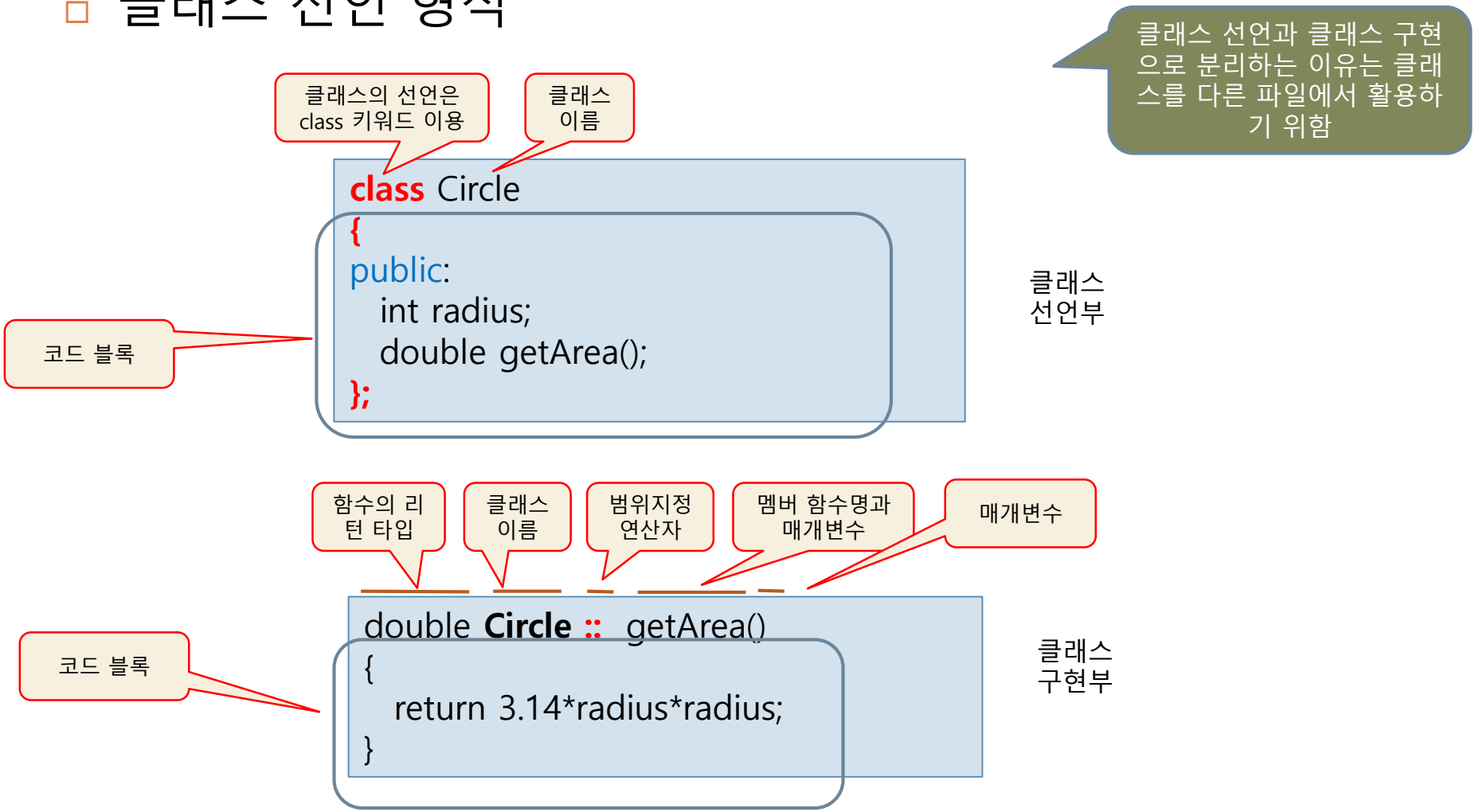
- 객체는 상태(state)와 행동(behavior)으로 구성
- TV 객체 사례
  - 상태
    - on/off 속성 – 현재 작동 중인지 표시
    - 채널(channel) – 현재 방송중인 채널
    - 음량(volume) – 현재 출력되는 소리 크기
  - 행동
    - 켜기(power on)
    - 끄기(power off)
    - 채널 증가(increase channel)
    - 채널 감소(decrease channel)
    - 음량 증가(increase volume)
    - 음량 줄이기(decrease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

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 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

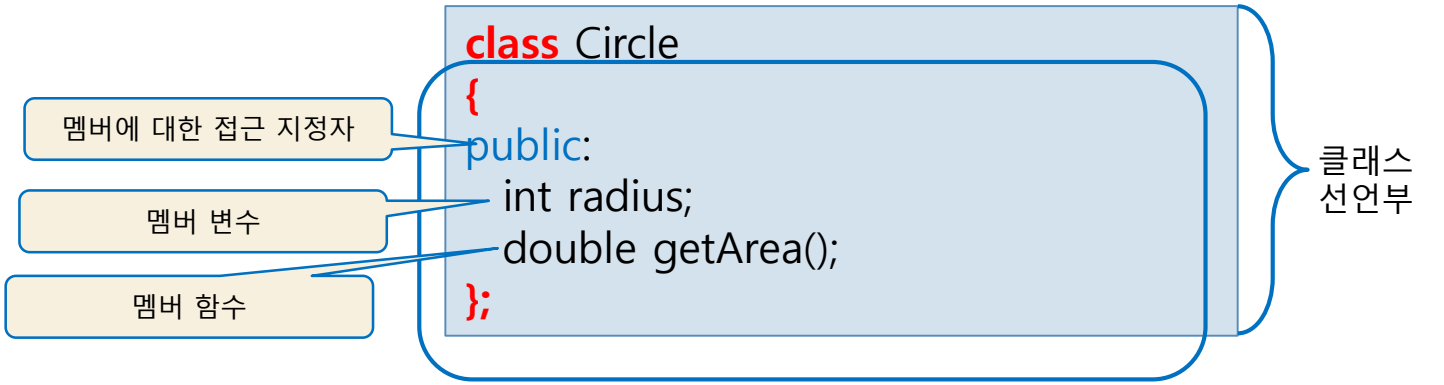
클래스 만들기 설명

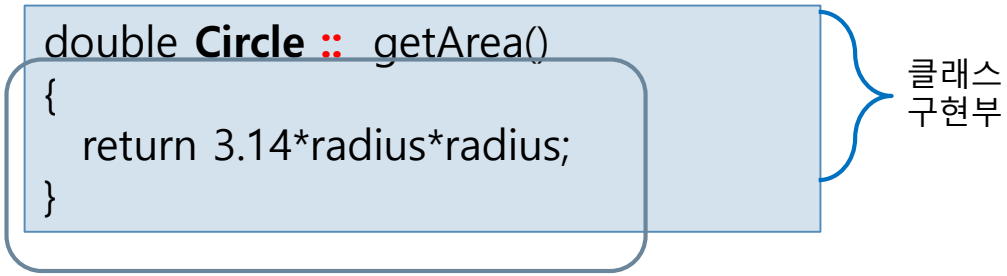
클래스 선언 형식



클래스 만들기 설명

- 클래스 선언 형식

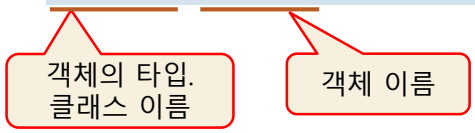




객체 생성 및 활용 설명

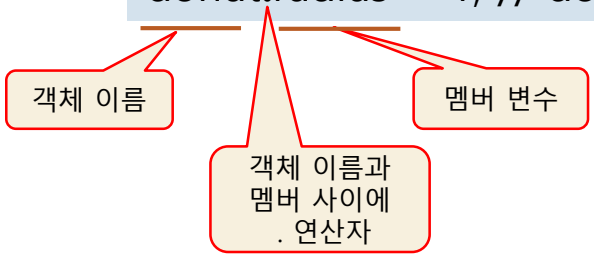
- 객체 이름 및 객체 생성

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



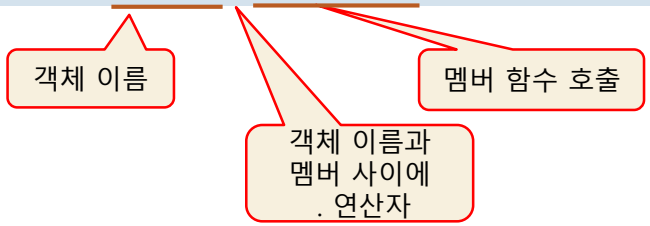
- 객체의 멤버 변수 접근

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



- 객체의 멤버 함수 접근

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



예제 - Circle 클래스의 객체 생성 및 활용

ch09_ex06.cpp

```
#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 area : " << area << endl;

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

1. 클래스 선언부

2. 클래스 구현부

3. 객체 생성

멤버 변수 접근

멤버 함수 호출

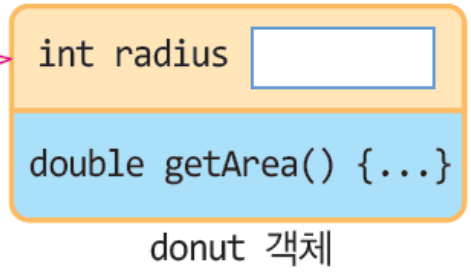
```
donut area : 3.14
pizza area : 2826
```

객체 이름과 생성, 접근 과정

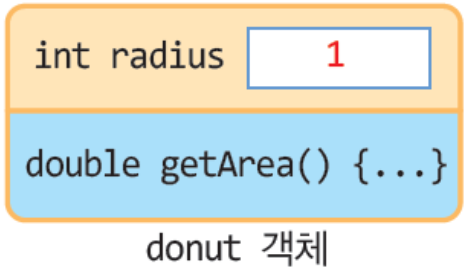
(1) Circle donut;

객체 이름

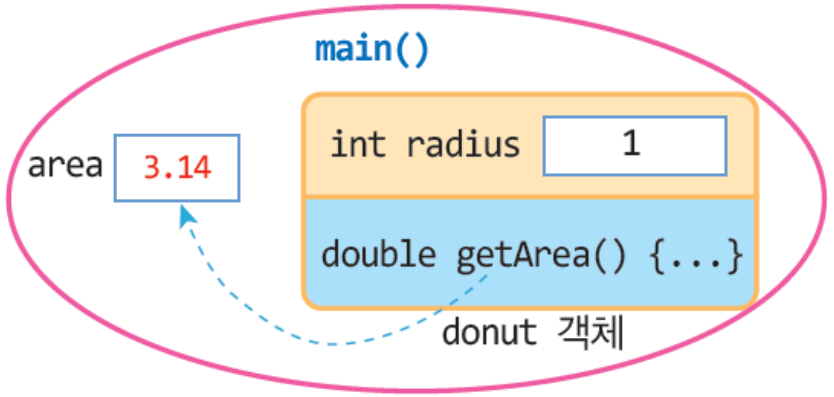
객체가 생성되면
메모리가 할당된다.



(2) donut.radius = 1;



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



문제 - Rectangle 클래스 만들기

- Rectangle 클래스를 만드세요
- Rectangle 클래스는 width , height 멤버변수를 가짐
- Rectangle 클래스는 다음과 같은 멤버 함수를 가짐
  - 리턴타입 int
  - 함수명 : getArea()
  - 매개변수 없음

ch09_ex07.cpp

```
int main() {
    Rectangle rect;
    rect.width = 3;
    rect.height = 5;
    cout << "사각형의 면적은 " << rect.getArea() << endl;
}
```

사각형의 면적은 15

문제 - Rectangle 클래스 만들기

ch09_ex07.cpp

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

class Rectangle { // Rectangle 클래스 선언
public:
   int width;
   int height;
   int getArea(); // 면적을 계산하여 리턴하는 함수
};

int Rectangle::getArea() {
   return width*height;
}

int main() {
   Rectangle rect;
   rect.width = 3;
   rect.height = 5;
   cout << "Area : " << rect.getArea() << endl;
}
```

1. 클래스 선언부

2. 클래스 구현부

3. 객체 생성

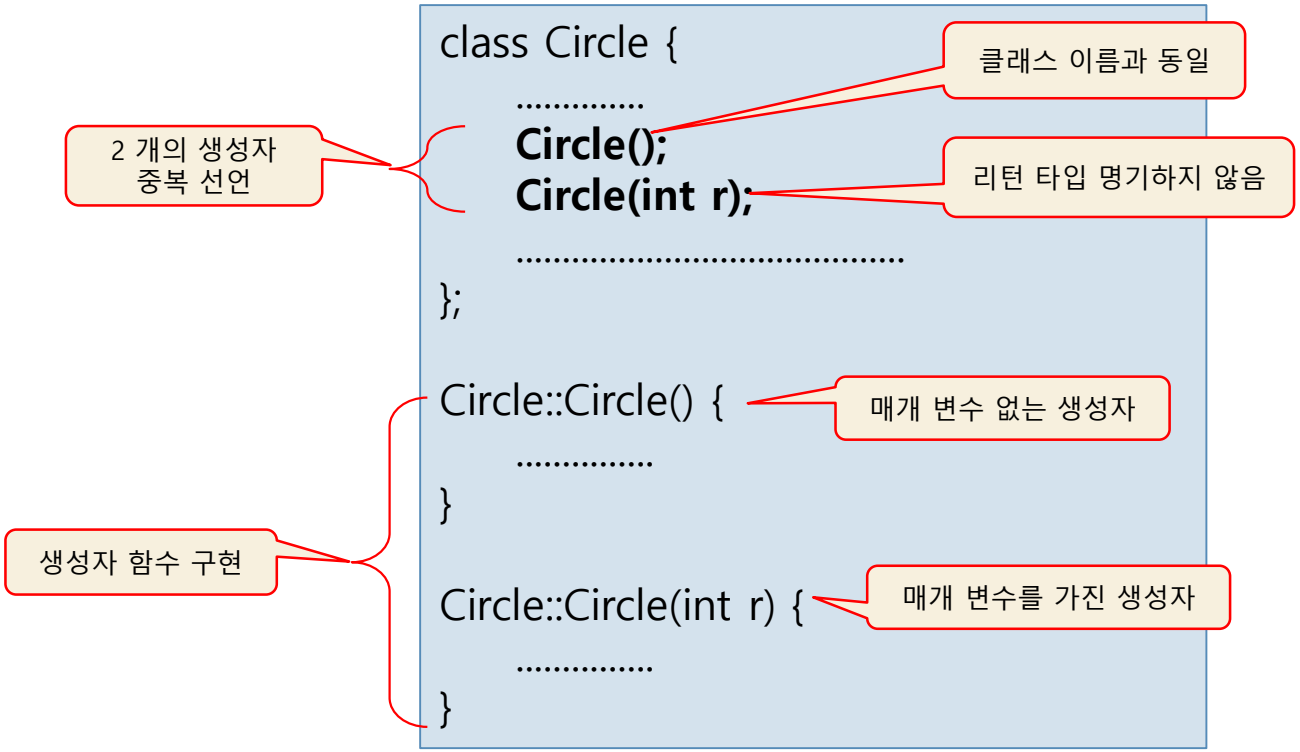
멤버 변수 접근

멤버 함수 호출

Area : 15

생성자

- 생성자(constructor)
  - 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
  - 클래스 이름과 동일한 멤버 함수
  - 함수와 다른점
    - 리턴 타입이 없음(void 도 없어야 함)
    - 함수명은 클래스명과 같아야 함



생성자 함수의 특징

- 생성자의 목적
  - 객체가 생성될 때 객체가 필요한 초기화를 위해
    - 멤버 변수 값 초기화, 메모리 할당, 파일 열기, 네트워크 연결 등
- 생성자 이름
  - 반드시 클래스 이름과 동일
- 객체 생성 시 오직 한 번만 호출
  - 자동으로 호출됨. 임의로 호출할 수 없음. 각 객체마다 생성자 실행
- 생성자는 중복 가능
  - 생성자는 한 클래스 내에 여러 개 가능
  - 중복된 생성자 중 하나만 실행
- 생성자가 선언되어 있지 않으면 기본 생성자 자동으로 생성
  - 기본 생성자 – 매개 변수 없는 생성자
  - 컴파일러에 의해 자동 생성

기본 생성자

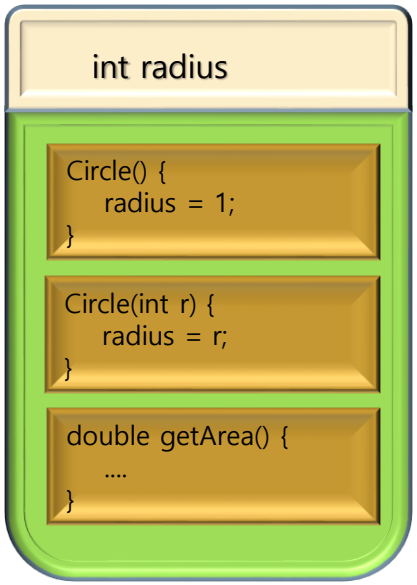
- 기본 생성자란?
  - 클래스에 생성자가 하나도 선언되어 있지 않은 경우, 컴파일러가 대신 삽입해주는 생성자
  - 매개 변수 없는 생성자
  - 디폴트 생성자라고도 부름

```
class Circle {
   .....
   Circle(); // 기본 생성자
};
```

```
class Circle{
   .....
   double getArea();
};
```

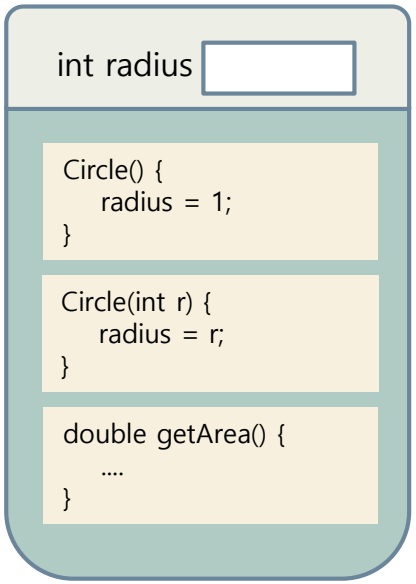
객체 생성 및 생성자 실행 과정

Circle donut;



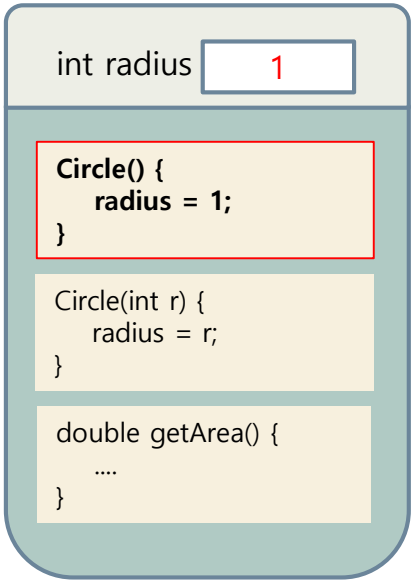
Circle 클래스

❶ 객체 공간 할당



donut 객체

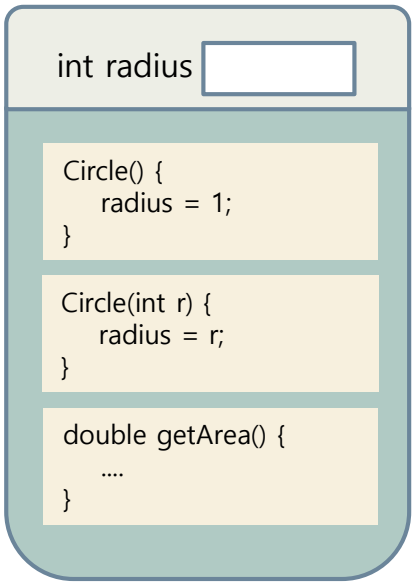
❷ 생성자 실행



donut 객체

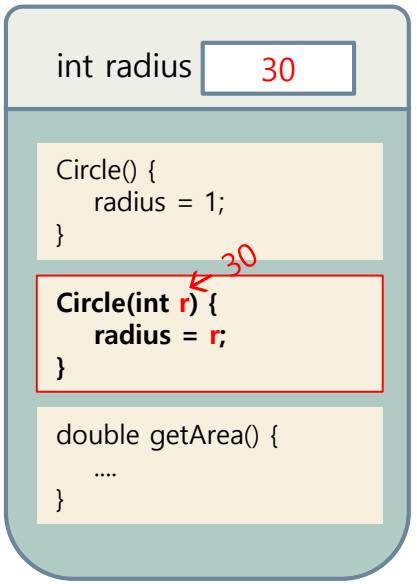
Circle pizza(30);

❶ 객체 공간 할당



pizza 객체

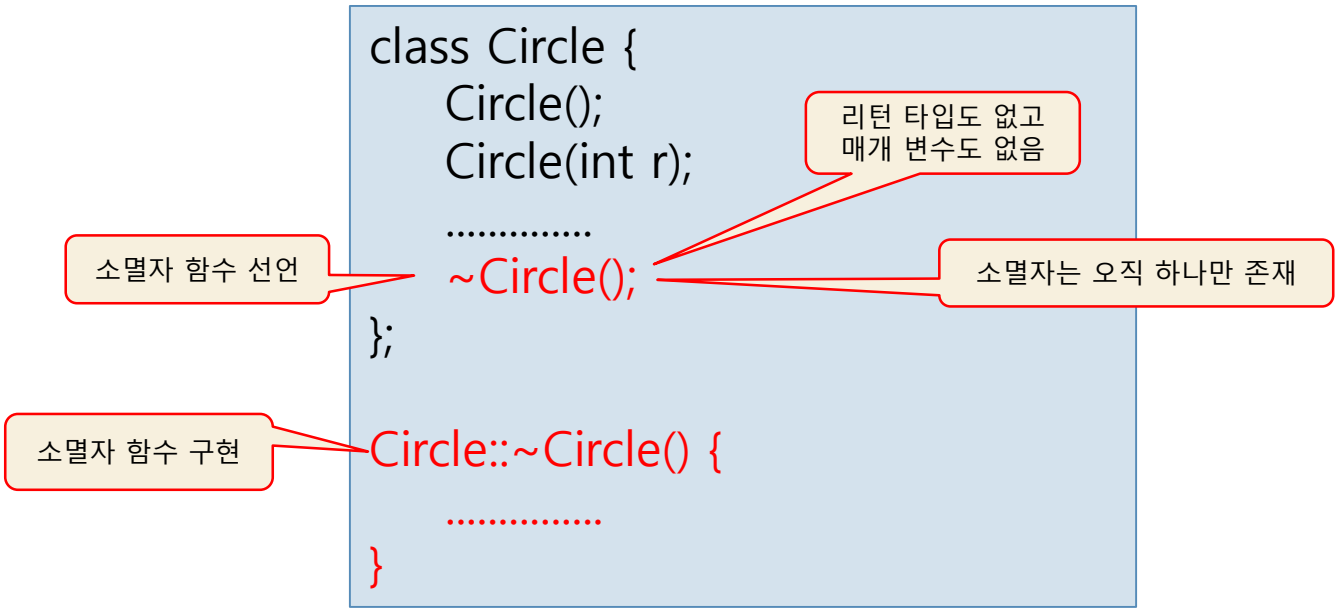
❷ 생성자 실행



pizza 객체

소멸자

- 소멸자
  - 객체가 소멸되는 시점에서 자동으로 호출되는 함수
    - 오직 한번만 자동 호출, 임의로 호출할 수 없음
    - 객체 메모리 소멸 직전 호출됨



소멸자 특징

- 소멸자의 목적
  - 객체가 사라질 때 마무리 작업을 위함
  - 실행 도중 동적으로 할당 받은 메모리 해제, 파일 저장 및 닫기, 네트워크 닫기 등
- 소멸자 함수의 이름은 클래스 이름 앞에 ~를 붙인다.
  - 예) Circle::~Circle() { ... }
- 소멸자는 리턴 타입이 없고, 어떤 값도 리턴하면 안됨
  - 리턴 타입 선언 불가
- 중복 불가능
  - 소멸자는 한 클래스 내에 오직 한 개만 작성 가능
  - 소멸자는 매개 변수 없는 함수
- 소멸자가 선언되어 있지 않으면 기본 소멸자가 자동 생성
  - 컴파일러에 의해 기본 소멸자 코드 생성
  - 컴파일러가 생성한 기본 소멸자 : 아무 것도 하지 않고 단순 리턴

ch09_ex08.cpp

자 작성 및 실행


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

class Circle {
public:
   int radius;
   Circle();
   Circle(int r);
   ~Circle(); // 소멸자
   double getArea();
};

Circle::Circle() {
   radius = 1;
   cout << "Circle() - radius " << radius << endl;
}

Circle::Circle(int r) {
   radius = r;
   cout << "Circle(int r) - radius " << radius  << endl;
}

Circle::~Circle() {
   cout << "~Circle() - radius " << radius <<  endl;
}

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

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

int main() {
   Circle donut;
   Circle pizza(30);

   return 0;
}
```

main() 함수가 종료하면 main() 함수의 스택에 생성된 pizza, donut 객체가 소멸된다.

```
Circle() - radius 1
Circle(int r) - radius 30
~Circle() - radius 30
~Circle() - radius 1
```

객체는 생성의 반대 순으로 소멸된다.

접근 지정자

- 캡슐화의 목적
  - 객체 보호, 보안
  - C++에서 객체의 캡슐화 전략
    - 객체의 상태를 나타내는 데이터 멤버(멤버 변수)에 대한 보호
    - 중요한 멤버는 다른 클래스나 객체에서 접근할 수 없도록 보호
    - 외부와의 인터페이스를 위해서 일부 멤버는 외부에 접근 허용

- 멤버에 대한 3 가지 접근 지정자
  - private
    - 동일한 클래스의 멤버 함수에만 제한함
  - public
    - 모든 다른 클래스에 허용
  - protected
    - 클래스 자신과 상속받은 자식 클래스에만 허용

```
class Sample {
private:
   // private 멤버 선언
public:
   // public 멤버 선언
protected:
   // protected 멤버 선언
};
```

접근 지정자


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


Circle::Circle() {
   radius = 1;
}
Circle::Circle(int r) {
   radius = r;
}
int main() {
   Circle waffle;
   waffle.radius = 5;
}
```

멤버 변수 보호받지 못함

노출된 멤버는 마음대로 접근. 나쁜 사례

(a) 멤버 변수를 public으로 선언한 나쁜 사례

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

Circle::Circle() {
   radius = 1;
}
Circle::Circle(int r) {
   radius = r;
}
int main() {
   Circle waffle(5); // 생성자에서 radius 설정
   waffle.radius = 5; // private 멤버 접근 불가
}
```

멤버 변수 보호받고 있음

(b) 멤버 변수를 private으로 선언한 바람직한 사례

예제 – 접근 지정자

ch09_ex09.cpp

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

class Circle {
   private:
   int radius;
public:
   double getArea();
   int getRadius();
   void setRadius(int r);
};

double Circle::getArea() {
   return 3.14*radius*radius;
}
void Circle::setRadius(int r) {
   radius = r;
}
int Circle::getRadius() {
   return radius;
}
```

```
int main() {
   int d_r = 1;
   Circle donut;
   donut.setRadius(d_r);
   double area = donut.getArea();
   cout << "donut area : " << area << endl;

   Circle pizza;
   int p_r = 30;
   pizza.setRadius(p_r);
   area = pizza.getArea();
   cout << "pizza area : " << area << endl;
}
```

```
donut area : 3.14
pizza area : 2826
```

바람직한 C++ 프로그램 작성법

- 클래스를 헤더 파일과 cpp 파일로 분리하여 작성
  - 클래스마다 분리 저장
  - 클래스 선언 부
    - 헤더 파일(.h)에 저장
  - 클래스 구현 부
    - cpp 파일에 저장
    - 클래스가 선언된 헤더 파일 include
  - main() 등 전역 함수나 변수는 다른 cpp 파일에 분산 저장
    - 필요하면 클래스가 선언된 헤더 파일 include
- 목적
  - 클래스 재사용

#include <헤더파일>와 #include "헤더파일"

- #include <헤더파일>
  - '헤더파일'을 찾는 위치
    - 컴파일러가 설치된 폴더에서 찾으라는 지시
    - 예) #include <iostream>은 iostream 파일을 컴파일러가 설치된 폴더에서 찾도록 지시

- #include "헤더파일"
  - '헤더파일'을 찾는 위치
    - 개발자의 프로젝트 폴더나
    - 개발자가 컴파일 옵션으로 지정한 include 폴더에서 찾도록 지시

ch09_ex08.cpp 의 소스를 헤더 파일과 cpp 파일로 분리하여 작성한 사례

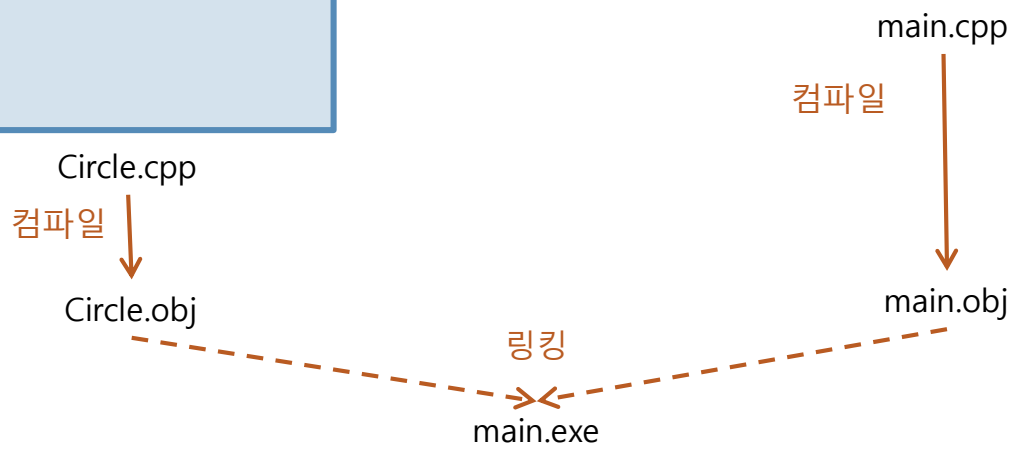
ch09_ex11

Circle.h

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

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

#include "Circle.h"

Circle::Circle() {
    radius = 1;
    cout << "Circle() - radius " << radius << endl;
}

Circle::Circle(int r) {
    radius = r;
    cout << "Circle(int r) - radius " << radius  << endl;
}

Circle::~Circle() {
    cout << "~Circle() - radius " << radius <<  endl;
}

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

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

#include "Circle.h"

int main() {
    Circle donu;
    Circle pizza(30);
    return 0;
}
```




```
Circle() - radius 1
Circle(int r) - radius 30
~Circle() - radius 30
~Circle() - radius 1
```

문제 - 헤더 파일과 cpp 파일로 분리하기

아래의 소스를 헤더 파일과 cpp 파일로 분리하여 재작성하라.

ch09_ex11

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

class Adder { // 덧셈 모듈 클래스
   int op1, op2;
public:
   Adder(int a, int b);
   int process();
};

Adder::Adder(int a, int b) {
   op1 = a; op2 = b;
}

int Adder::process() {
   return op1 + op2;
}
```

```
class Calculator { // 계산기 클래스
public:
   void run();
};

void Calculator::run() {
   cout << "두 개의 수를 입력하세요>>";
   int a, b;
   cin >> a >> b; // 정수 두 개 입력
   Adder adder(a, b); // 덧셈기 생성
   cout << adder.process(); // 덧셈 계산
}

int main() {
   Calculator calc; // calc 객체 생성
   calc.run(); // 계산기 시작
}
```

```
두 개의 수를 입력하세요>>5 -20
-15
```

문제 - 헤더 파일과 cpp 파일로 분리하기

Adder.h

```
#ifndef ADDER_H
#define ADDER_H

class Adder { // 덧셈 모듈 클래스
    int op1, op2;
public:
    Adder(int a, int b);
    int process();
};

#endif
```

Calculator.h

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

class Calculator { // 계산기 클래스
public:
    void run();
};

#endif
```

ch09_ex_07

Adder.cpp

```
#include "Adder.h"

Adder::Adder(int a, int b) {
    op1 = a; op2 = b;
}

int Adder::process() {
    return op1 + op2;
}
```

Calculator.cpp

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

#include "Calculator.h"
#include "Adder.h"

void Calculator::run() {
    cout << "두 개의 수를 입력하세요>>";
    int a, b;
    cin >> a >> b; // 정수 두 개 입력
    Adder adder(a, b); // 덧셈기 생성
    cout << adder.process(); // 덧셈 계산
}
```

main.cpp

```
#include "Calculator.h"

int main() {
    Calculator calc; // calc 객체 생성
    calc.run(); // 계산기 시작
}
```

```
두 개의 수를 입력하세요>>5 -20
-15
```