ROVIN 2010 Robot Vision & Intelligence Lab.

고급프로그래밍

클래스와 객체

정문호 교수
로봇 비전 및 지능 연구실
광운대학교
(02-940-5625, mhjeong@kw.ac.kr)

Schedule

week	Topics		Homework	Quiz
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본(3), 클래스와 객체	1	1
3		객체생성과 사용, 함수와 참조	2	2
4		복사 생성자와 함수중복, static, friend, 연산자 중복	3	3
5		상속, 가상함수와 추상클래스	4	4
6		템플릿과 STL, 표준 입출력	5	5
7		파일 입출력		
8				
9	C++	예외처리 및 C 사용, 람다식	6	6
10		멀티스레딩	7	7
11		멀티스레딩, 고급문법	8	8
12		고급문법	9	9
13	병렬 프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15	기말고사			

오늘의 학습내용

- C++ 클래스와 객체

클래스와 객체

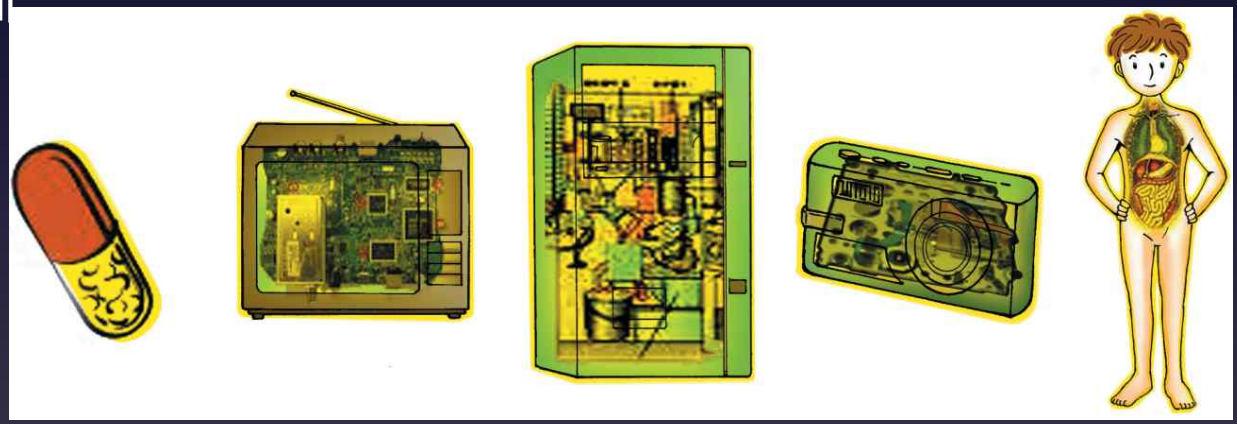
세상의 모든 것이 객체이다.

- 세상 모든 것이 객체

TV 의자 책 집 카메라 컴퓨터

객체는 캡슐화된다.

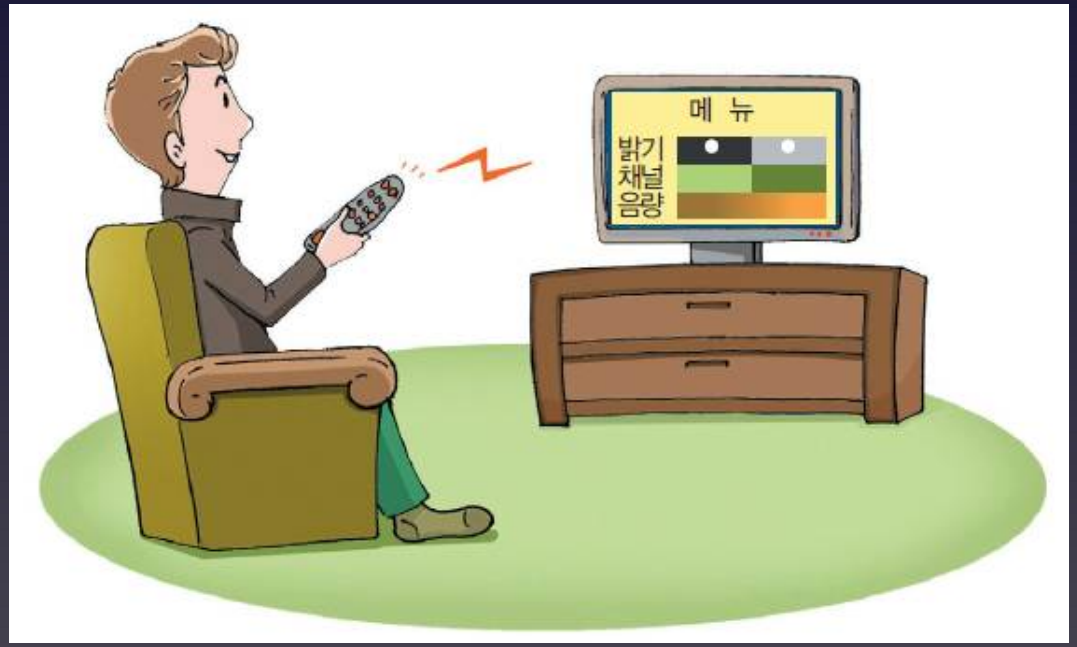
- 캡슐화(encapsulation)
  - 객체의 본질적인 특성
  - 객체를 캡슐로 싸서 그 내부를 보호하고 볼 수 없게 함
    - 캡슐에 든 약은 어떤 색인지 어떤 성분인지 보이지 않고, 외부로부터 안전
  - 캡슐화 사례



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

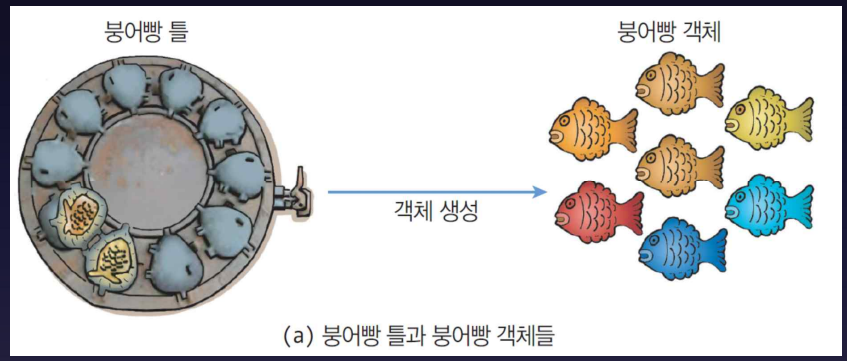
객체의 일부 요소는 공개된다.

- 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++클래스와 C++객체

- 클래스
  - 객체를 만들어내기 위해 정의된 설계도, 틀
  - 클래스는 객체가 아님. 실체도 아님
  - 멤버 변수와 멤버 함수 선언



(a) 붕어빵 틀과 붕어빵 객체들

- 객체
  - 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

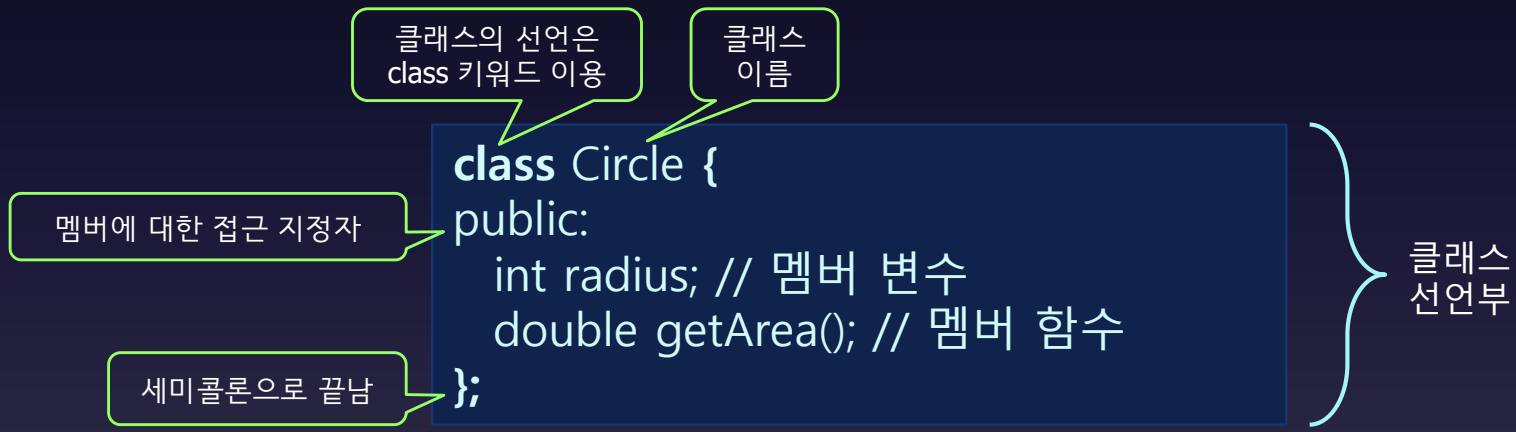
C++ 클래스 만들기

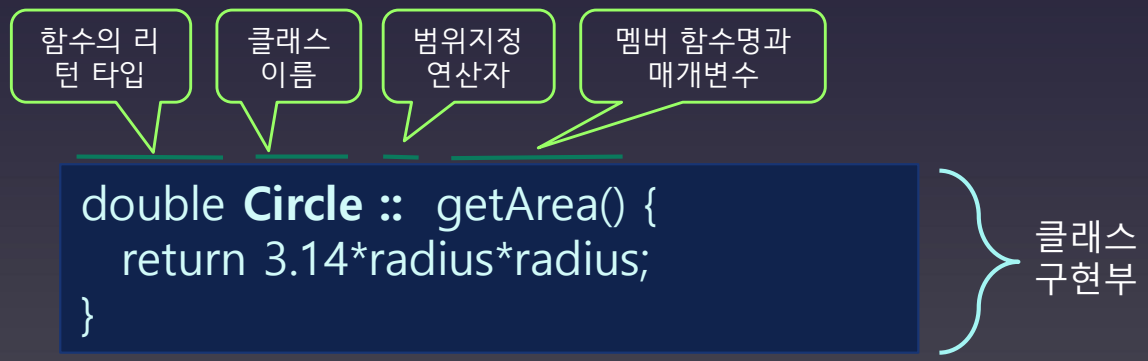
- 클래스 작성
  - 멤버 변수와 멤버 함수로 구성
  - 클래스 선언부와 클래스 구현 (정의) 부로 구성

- 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 선언된 모든 멤버 함수 구현

클래스 만들기 설명

- 클래스 선언과 클래스 구현으로 분리하는 이유는 클래스를 다른 파일에서 활용하기 위함





예제

Circle 선언부

Circle 구현부

객체 donut 생성

멤버 변수 접근

멤버 함수 호출

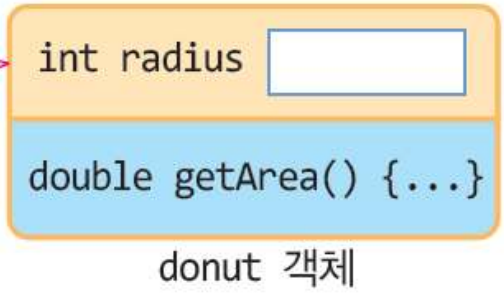
```
#include <iostream>
using namespace std;

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;
}
```

객체 이름과 생성, 접근 과정

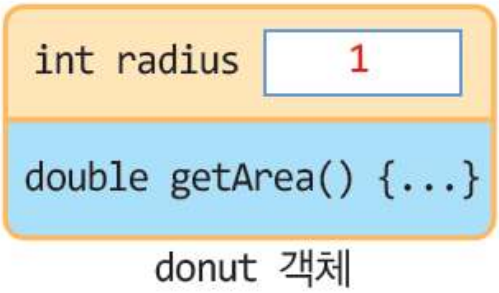
(1) Circle donut;

객체 이름

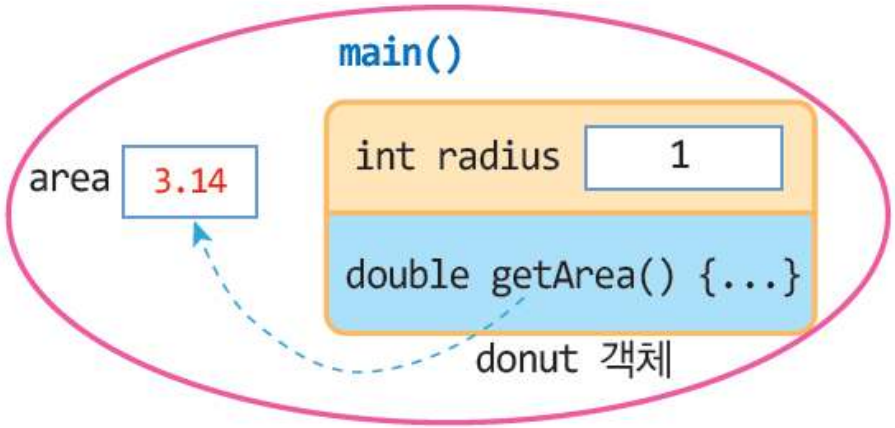
객체가 생성되면
메모리가 할당된다.



(2) donut.radius = 1;



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



예제 – Rectangle 클래스 만들기

- 다음 main() 함수가 잘 작동하도록 너비(width)와 높이(height)를 가지고 면적 계산 기능을 가진Rectangle 클래스를 작성하고 전체 프로그램을 완성하라.

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

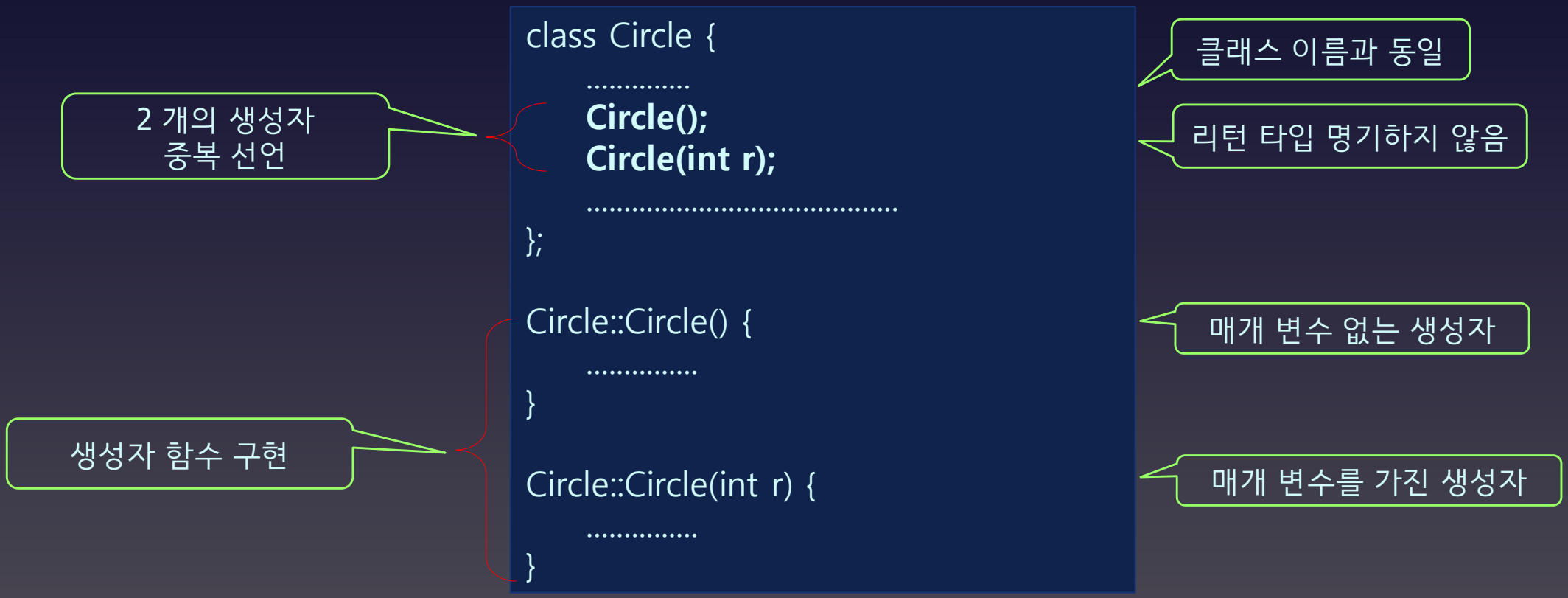
```
사각형의 면적은 15
```

예제 – Rectangle 클래스 만들기

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

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

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

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

생성자

- 생성자(constructor)
  - 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
  - 클래스 이름과 동일한 멤버 함수



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
- 생성자는 리턴 타입을 선언하지 않는다
  - 리턴 타입 없음. void 타입도 안됨
- 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

예제 - 2 개의 생성자

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

class Circle {
public:
   int radius;
   Circle(); // 매개 변수 없는 생성자
   Circle(int r); // 매개 변수 있는 생성자
   double getArea();
};

Circle::Circle() {
   radius = 1;
   cout << "반지름 " << radius << " 원 생성"
        << endl;
}

Circle::Circle(int r) {
   radius = r;
   cout << "반지름 " << radius << " 원 생성"
        << endl;
}

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

Circle(); 자동 호출

Circle(30); 자동 호출

```
int main() {
   Circle donut; // 매개 변수 없는 생성자 호출
   double area = donut.getArea();
   cout << "donut 면적은 " << area << endl;

   Circle pizza(30); // 매개 변수 있는 생성자 호출
   area = pizza.getArea();
   cout << "pizza 면적은 " << area << endl;
}
```

```
반지름 1 원 생성
donut 면적은 3.14
반지름 30 원 생성
pizza 면적은 2826
```

위임 생성자

- 여러 생성자에 중복 작성된 코드의 간소화
  - 타겟 생성자와 이를 호출하는 위임 생성자로 나누어 작성
    - 타겟 생성자 : 객체 초기화를 전담하는 생성자
    - 위임 생성자 : 타겟 생성자를 호출하는 생성자, 객체 초기화를 타겟 생성자에 위임

```
Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius << " 원 생성" << endl;
}
Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius << " 원 생성" << endl;
}
```

여러 생성자에 코드 중복

위임 생성자

타겟 생성자

```
Circle::Circle() : Circle(1) { } // Circle(int r)의 생성자 호출

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius << " 원 생성" << endl;
}
```

호출

간소화된 코드

예제 – 위임 생성자 만들기

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

class Circle {
public:
    int radius;
    Circle(); // 위임 생성자
    Circle(int r); // 타겟 생성자
    double getArea();
};

Circle::Circle() : Circle(1) { } // 위임 생성자



Circle::Circle(int r) { // 타겟 생성자
    radius = r;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

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

```
int main() {
    Circle donut; // 매개 변수 없는 생성자 호출
    double area = donut.getArea();
    cout << "donut 면적은 " << area << endl;

    Circle pizza(30); // 매개 변수 있는 생성자 호출
    area = pizza.getArea();
    cout << "pizza 면적은 " << area << endl;
}
```

```
반지름 1 원 생성
donut 면적은 3.14
반지름 30 원 생성
pizza 면적은 2826
```

멤버 변수 초기화 방법

```
class Point {
    int x, y;

public:
    Point();
    Point(int a, int b);
};
```

(1) 생성자 코드에서 멤버 변수 초기화

```
Point::Point() { x = 0; y = 0; }
Point::Point(int a, int b) { x = a; y = b; }
```

(2) 생성자 서두에 초기값으로 초기화

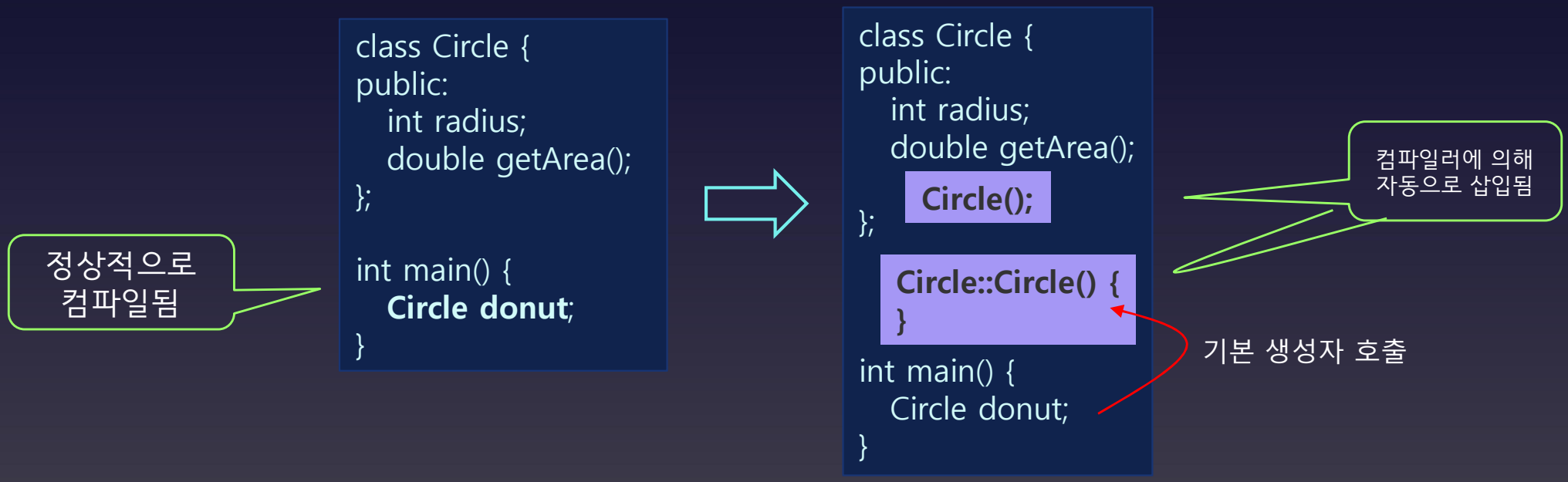
```
Point::Point() : x(0), y(0) { // 멤버 변수 x, y를 0으로 초기화
}
Point::Point(int a, int b)    // 멤버 변수 x=a로, y=b로 초기화
   : x(a), y(b) {             // 콜론(:) 이하 부분을 밑줄에 써도 됨
}
```

(3) 클래스 선언부에서 직접 초기화

```
class Point {
   int x=0; y=0; // 클래스 선언부에서 x, y를 0으로 초기화
public:
   ...
};
```

기본 생성자의 자동 생성

- 생성자가 하나도 작성되어 있지 않은 클래스의 경우
  - 컴파일러가 기본 (디폴트) 생성자 자동 생성



(a) 생성자를 선언하지 않는 Circle 클래스

(b) 컴파일러에 의해 기본 생성자 자동 삽입

기본 생성자의 자동 생성(X)

- 생성자가 하나라도 선언된 클래스의 경우
  - 컴파일러는 기본 생성자를 자동 생성하지 않음

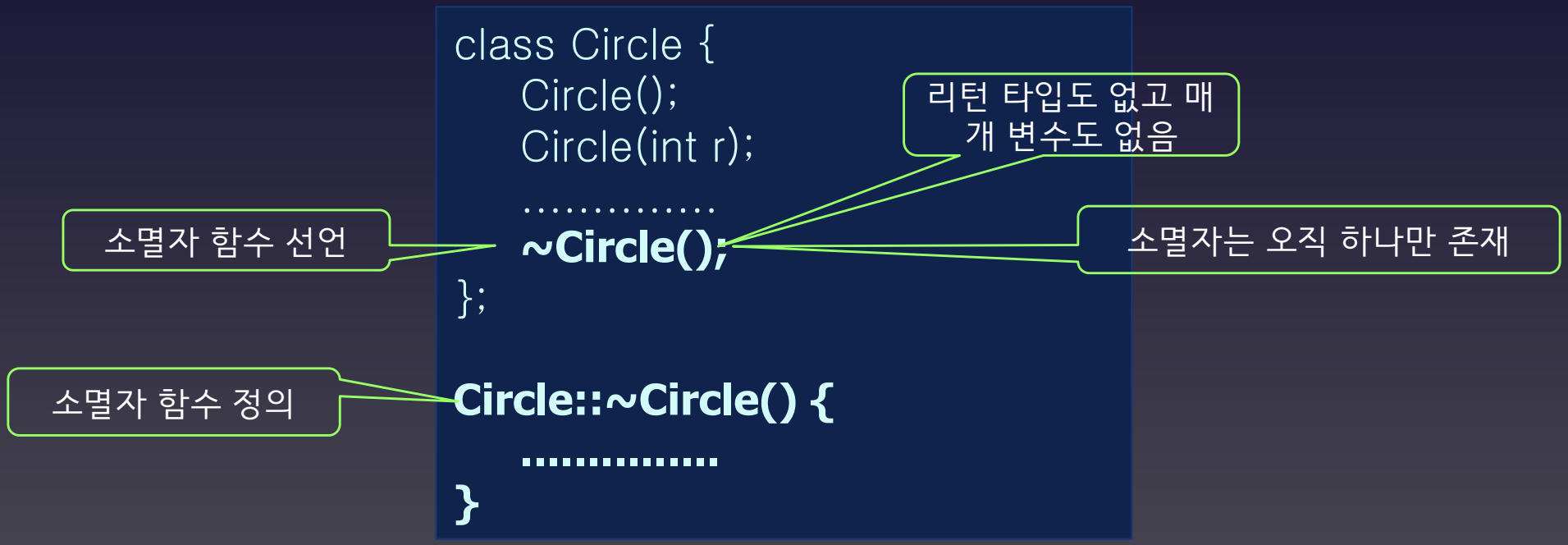
```
class Circle {
public:
   int radius;
   double getArea();
   Circle(int r);
};

Circle::Circle(int r) {
   radius = r;
}

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

Circle 클래스에 생성자가 선언되어 있기 때문에, 컴파일러는 기본 생성자를 자동 생성하지 않음

컴파일 오류
기본 생성자 없음

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

예제

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

class Circle {
public:
   int radius;

   Circle();
   Circle(int r);
   ~Circle(); // 소멸자
   double getArea();
};


Circle::Circle() {
   radius = 1;
   cout << "반지름 " << radius << " 원 생성"
        << endl;
}
```

```
Circle::Circle(int r) {
   radius = r;
   cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::~Circle(){
   cout << "반지름 " << radius << " 원 소멸" << endl;
}

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.

```
반지름 1 원 생성
반지름 30 원 생성
반지름 30 원 소멸
반지름 1 원 소멸
```

객체는 생성의 반대순으로 소멸된다.

생성자/소멸자 실행 순서

- 객체가 선언된 위치에 따른 분류
  - 지역 객체
    - 함수 내에 선언된 객체로서, 함수가 종료하면 소멸된다.
  - 전역 객체
    - 함수의 바깥에 선언된 객체로서, 프로그램이 종료할 때 소멸된다.

- 객체 생성 순서
  - 전역 객체는 프로그램에 선언된 순서로 생성
  - 지역 객체는 함수가 호출되는 순간에 순서대로 생성
- 객체 소멸 순서
  - 함수가 종료하면, 지역 객체가 생성된 순서의 역순으로 소멸
  - 프로그램이 종료하면, 전역 객체가 생성된 순서의 역순으로 소멸

- new를 이용하여 동적으로 생성된 객체의 경우
  - new를 실행하는 순간 객체 생성
  - delete 연산자를 실행할 때 객체 소멸

예제 - 생성 및 소멸 순서

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

Circle::Circle() {
    radius = 1;
    cout<< "반지름 " << radius<< "원 생성"<< endl;
}
Circle::Circle(int r) {
    radius = r;
    cout<< "반지름 " << radius<< "원 생성"<< endl;
}

Circle::~Circle() {
    cout<< "반지름 " << radius<< "원 소멸"<< endl;
}
double Circle::getArea() {
    return 3.14*radius*radius;
}
```

```
Circle globalDonut(1000);
Circle globalPizza(2000);

void f() {
    Circle fDonut(100);
    Circle fPizza(200);
}

int main() {
    Circle mainDonut;
    Circle mainPizza(30);
    f();
}
```

전역 객체 생성

지역 객체 생성

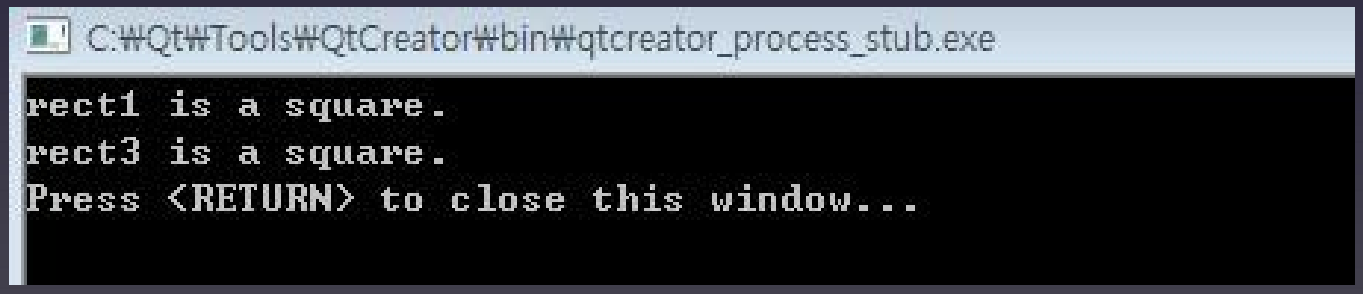
지역 객체 생성

```
반지름 1000 원 생성
반지름 2000 원 생성
반지름 1 원 생성
반지름 30 원 생성
반지름 100 원 생성
반지름 200 원 생성
반지름 200 원 소멸
반지름 100 원 소멸
반지름 30 원 소멸
반지름 1 원 소멸
반지름 2000 원 소멸
반지름 1000 원 소멸
```

실습 1 – Rectangle 클래스 만들기

- 다음 main() 함수가 잘 작동하도록 Rectangle 클래스를 작성하고 프로그램을 완성하라.
  - Rectangle 클래스는 width와 height의 두 멤버 변수와 3 개의 생성자, 그리고 isSquare() 함수를 가진다

```
int main()
{
    Rectangle rect1;
    Rectangle rect2(3, 5);
    Rectangle rect3(3);

    if(rect1.isSquare()) cout << "rect1 is a square." << endl;
    if(rect2.isSquare()) cout << "rect2 is a square." << endl;
    if(rect3.isSquare()) cout << "rect3 is a square." << endl;
}
```



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

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

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

Access specifiers	Inside the class	Outside the class
private	o	x
protected	o	x
public	o	o

중복 접근 지정과 디폴트 접근 지정

접근 지정의 중복 사용 가능

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

⟹

접근 지정의 중복 사례

```
class Sample {
private:
   int x, y;
public:
   Sample();
private:
   bool checkXY();
};
```

디폴트 접근 지정은 private

디폴트 접근 지정은 private

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

〓

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

멤버변수의 private 지정

```
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

실습 2 – 은행계좌 클래스

- 은행계좌를 표현하는 Account 클래스를 작성하시오. Account는 계좌의 주인 이름, 계좌번호, 잔액을 나타내는 세 개의 맴버 변수로 이루어지고, 다음과 같은 실행결과가 나와야 한다(단, 인라인 함수

```
int main()
{
    Account a("kitae", 1, 5000);    // id 1번, 잔액 5000원, 이름이 kitae인 계좌 생성
    a.deposit(50000);               // 50000원 저금
    cout << a.getOwner() << "'s ballance : " << a.inquiry() << endl;
    int money = a.withdraw(20000);  // 20000원 출금. withdraw()는 출금한 실제 금액 리턴
    cout << a.getOwner() << "'s ballance : " << a.inquiry() << endl;
}
```

```
C:₩Qt₩Tools₩QtCreator₩bin₩qtcreator_process_stub.exe
kitae's ballance : 55000
kitae's ballance : 35000
Press <RETURN> to close this window...
```