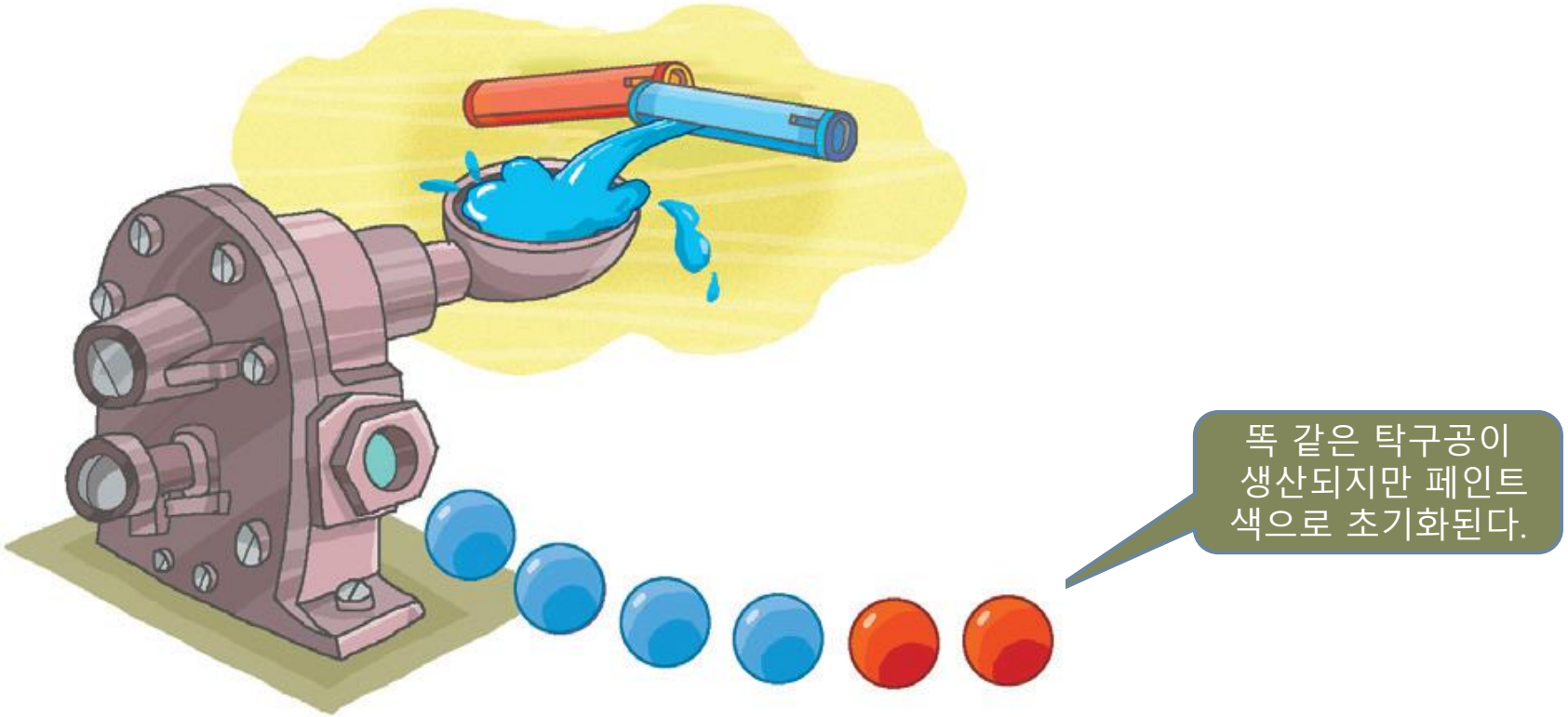
05. 생성자와 소멸자

5.1 생성자와 소멸자

5.2 분할컴파일

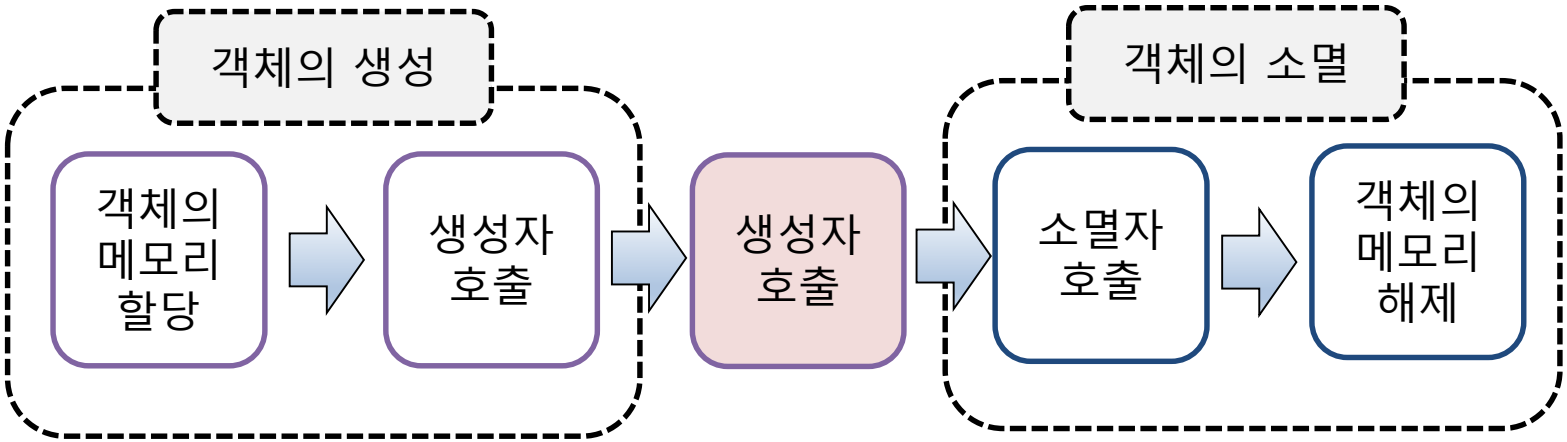
5.1 생성자와 소멸자

탁구공 생산 장치와 생성자



객체의 생성 및 소멸

- 객체의 생성
  - 객체가 생성될 때 생성자가 자동으로 호출된다.
- 객체의 소멸
  - 객체가 소멸될 때 소멸자가 자동으로 호출된다.



생성자와 소멸자

- 생성자와 소멸자는 자동으로 호출되는 함수이다.
  - 생성자 : 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
  - 소멸자 : 객체를 소멸되는 시점에서 자동으로 호출되는 멤버 함수

- 생성자와 소멸자 함수의 사용
  - 생성자 : 객체가 생성될 때 객체가 필요한 초기화를 위해
  - 소멸자 : 객체가 소멸될 때 객체의 정리를 위해

- 생성자와 소멸자 이름
  - 생성자 : **"클래스이름()"** 형식의 함수
  - 소멸자 : **"~클래스이름()"** 형식의 함수

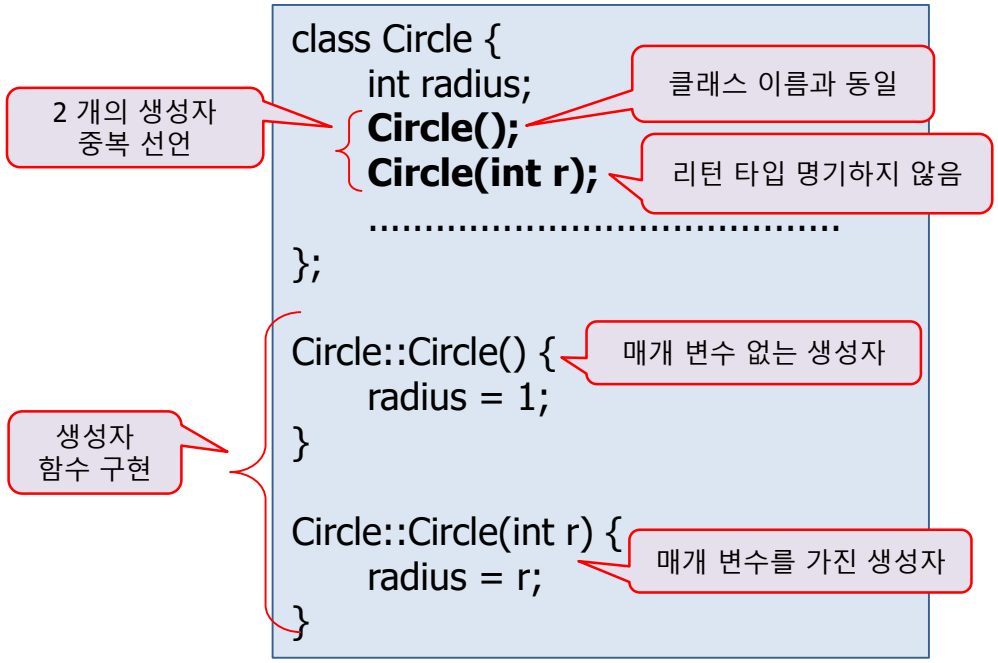
- 생성자와 소멸자는 리턴 값이 없음
  - 리턴 타입 없음. void도 안됨

생성자 특징

생성자 특징

- 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



string 클래스의 생성자 중복 사례

```
class string {
   .....
public:
   string(); // 빈 문자열을 가진 스트링 객체 생성
   string(string& str); // str을 복사한 새로운 스트링 객체 생성
   string(char* s); // '\0'로 끝나는 C-스트링 s를 스트링 객체로 생성
   .....
};
```

```
string str; // 빈 문자열을 가진 스트링 객체
string address("청주시 서원구 충대로1");
string copyAddress(address); // address의 문자열을 복사한 별도의 copyAddress 생성
```

2개의 생성자를 가진 Circle 클래스

```
#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

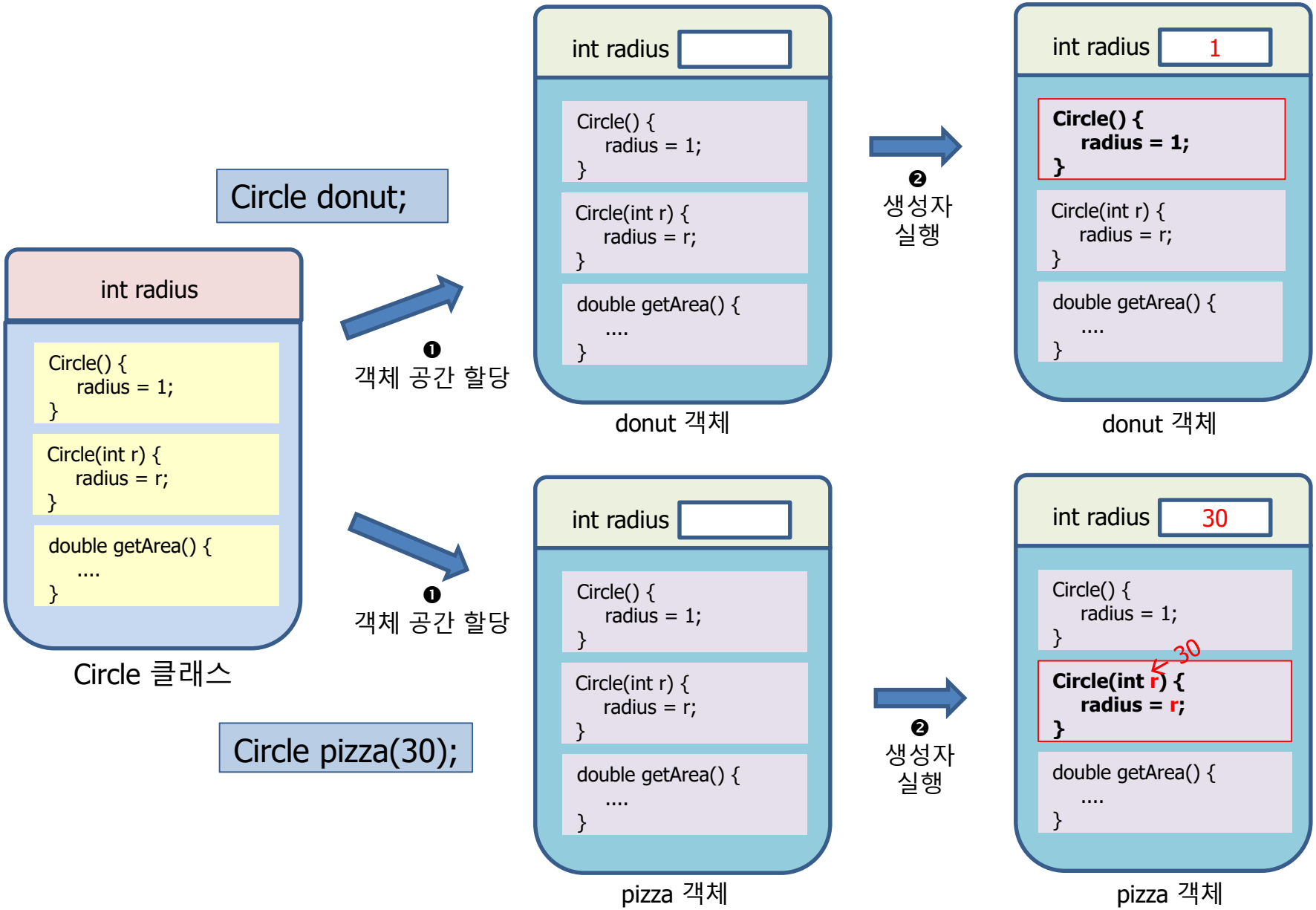
30

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
```

객체 생성 및 생성자 실행 과정



디폴트 생성자

1. *생성자는 꼭 있어야 하는가?*
   - 예. C++ 컴파일러는 객체가 생성될 때, 생성자 반드시 호출
2. *개발자가 클래스에 생성자를 작성해 놓지 않으면?*
   - 컴파일러에 의해 기본 생성자가 자동으로 생성

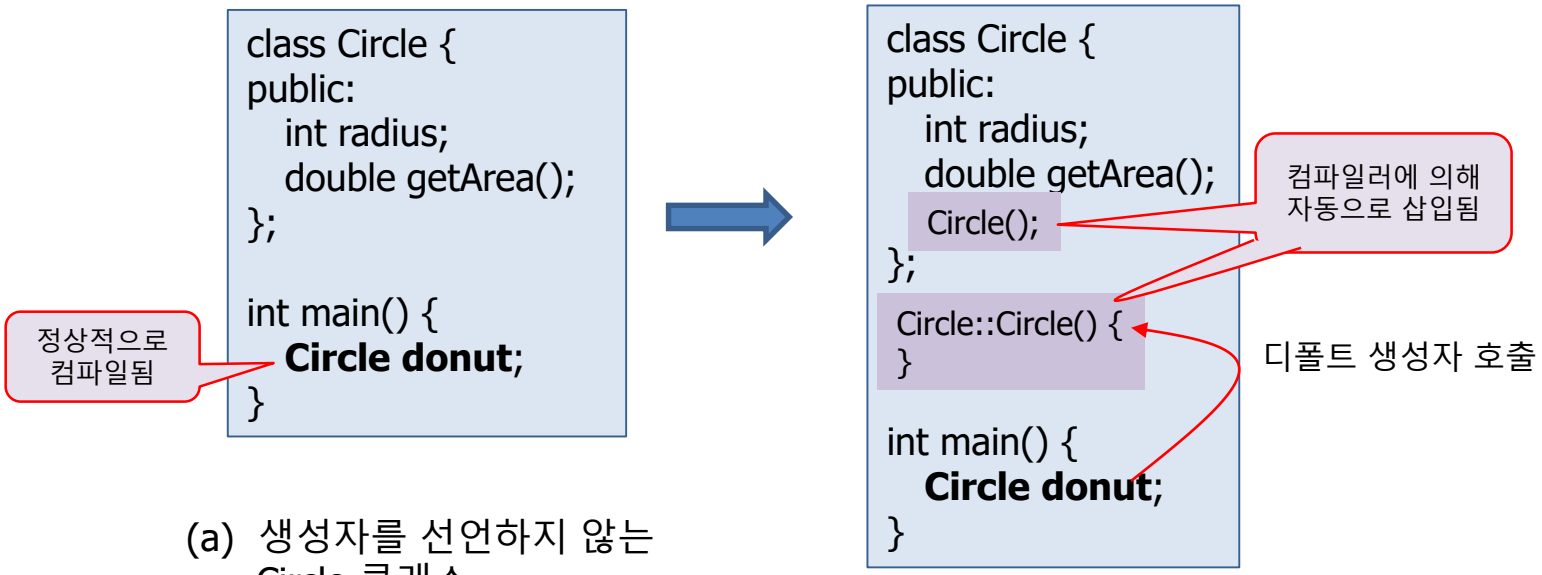
디폴트 생성자란?

- 매개 변수 없는 생성자. 기본 생성자라고도 부름.
- 객체를 생성할 때 별도로 지정하지 않으면 항상 디폴트 생성자로 초기화 됨.

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

디폴트 생성자가 자동으로 생성되는 경우

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

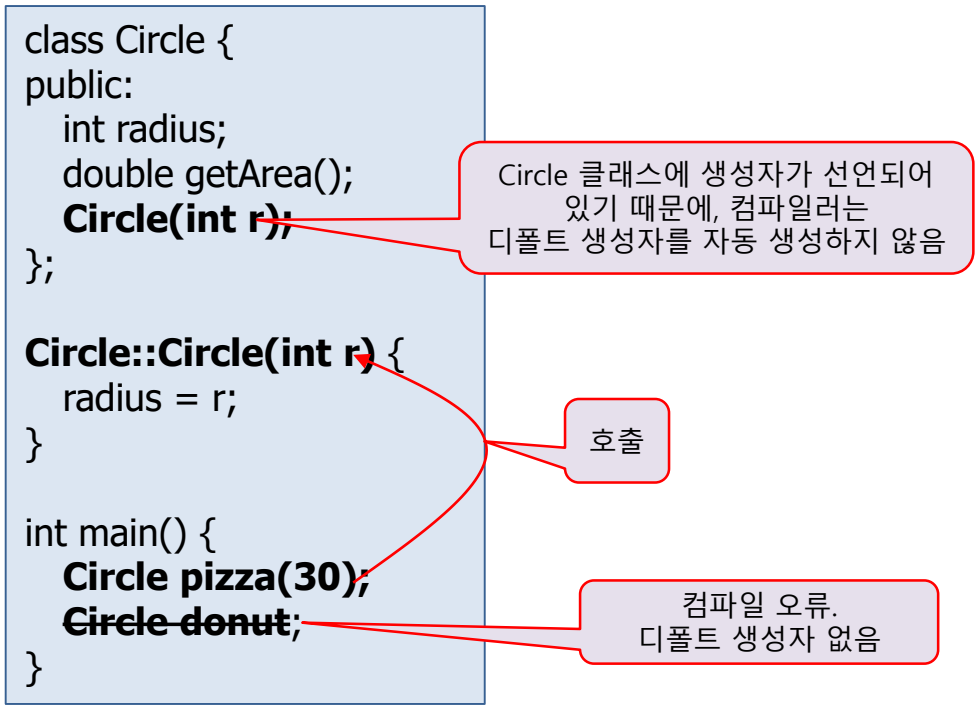


(a) 생성자를 선언하지 않는 Circle 클래스

(b) 컴파일러에 의해 디폴트 생성자 자동 삽입

디폴트 생성자가 자동으로 생성되지 않는 경우

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



디폴트 인자를 사용한 함수 중복 간소화

- 디폴트 인자의 장점 – 함수 중복 간소화

```
class Circle {
   ....................
public:
   Circle() { radius = 1; }
   Circle(int r) { radius = r;
}
   ....................
};
```

```
class Circle {
   ....................
public:
   Circle(int r=1) { radius = r;
   }
   ....................
};
```

2 개의 생성자 함수를
디폴트 매개 변수를 가진 하나의
함수로 간소화

- 중복 함수들과 디폴트 매개 변수를 가진 함수를 함께 사용 불가

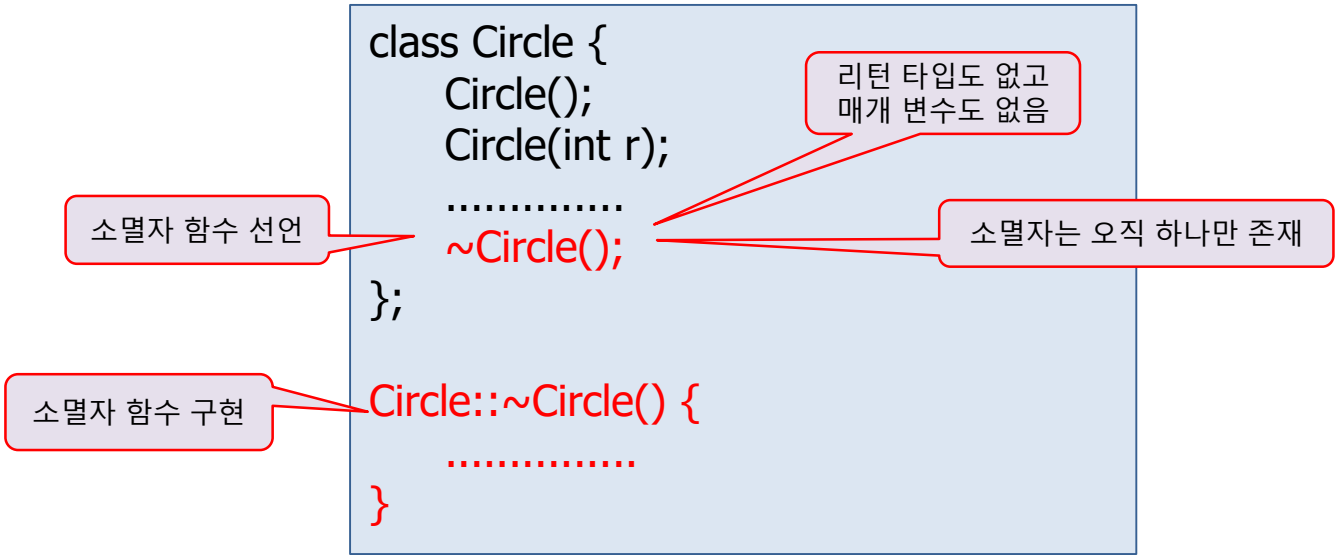
```
class Circle {
   ....................
public:
   Circle() { radius = 1; }
   Circle(int r) { radius = r; }
   Circle(int r=1) { radius = r;   }
   ....................
};
```

중복된 함수와
동시 사용 불가

소멸자

소멸자

- 객체가 소멸되는 시점에서 자동으로 호출되는 함수
  - 오직 한번만 자동 호출, 임의로 호출할 수 없음
  - 객체 메모리 소멸 직전 호출됨
- "**~클래스 이름()** " 형식의 함수
- 인자를 가질 수 없기 때문에 **오버로딩할 수 없음**



소멸자 특징

- 소멸자의 사용
  - 객체가 사라질 때 마무리 작업을 위함
  - 실행 도중 동적으로 할당 받은 메모리 해제, 파일 저장 및 닫기, 네트워크 닫기 등
- 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 디폴트(기본) 소멸자가 자동 생성
  - 컴파일러에 의해 디폴트 소멸자 코드 생성
  - 컴파일러가 생성한 디폴트 소멸자 : 아무 것도 하지 않고 단순 리턴

Circle 클래스에 소멸자 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 << "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

지역 객체와 전역 객체의 생성 및 소멸 순서

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

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;
}
```

다음 프로그램의 실행 결과는 무엇인가?

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

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

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

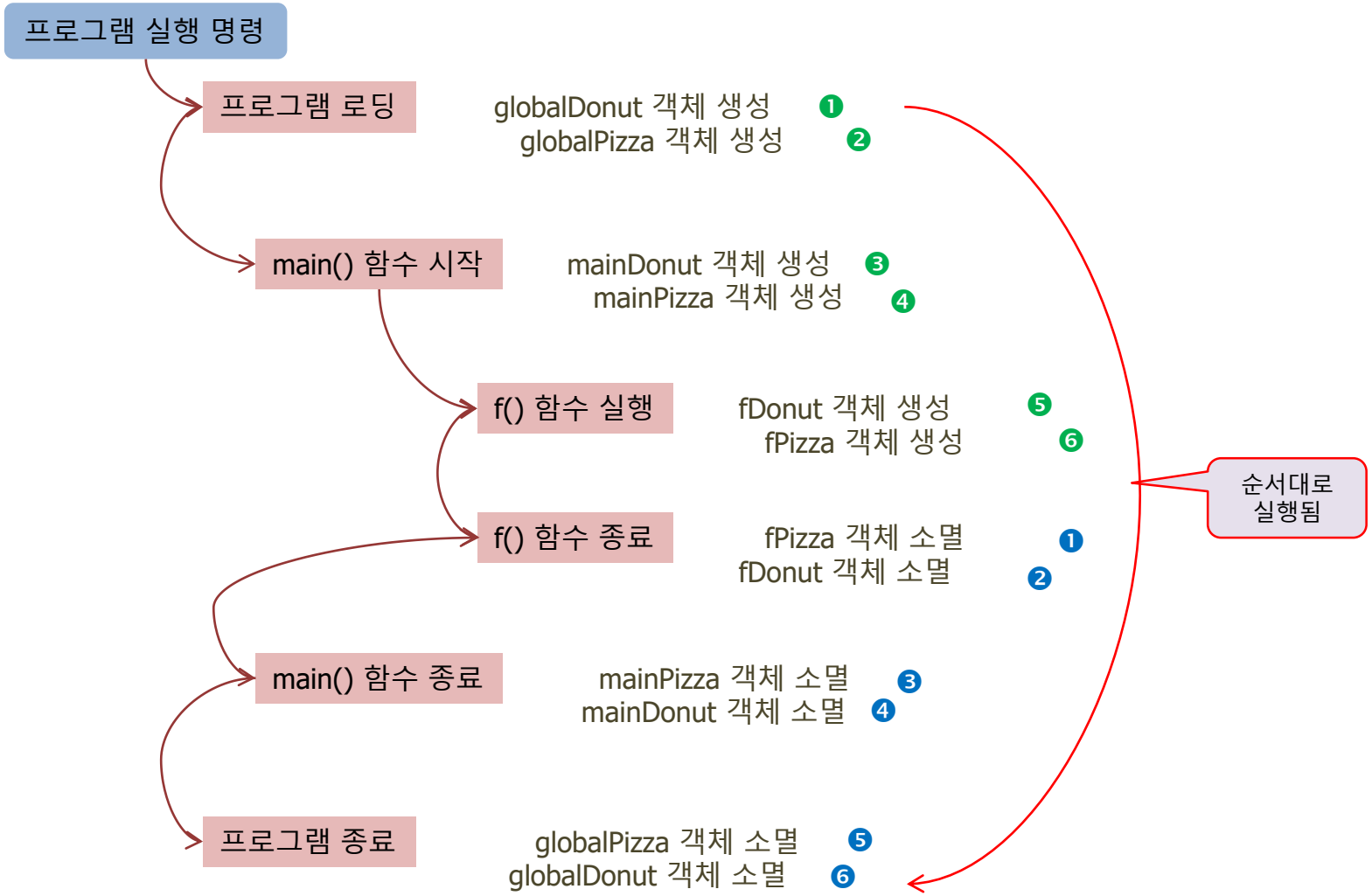
전역 객체 생성

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
```

지역 객체와 전역 객체의 생성 및 소멸 순서



객체 간의 초기화와 대입

- 같은 클래스의 객체 간에 서로 초기화나 대입이 가능

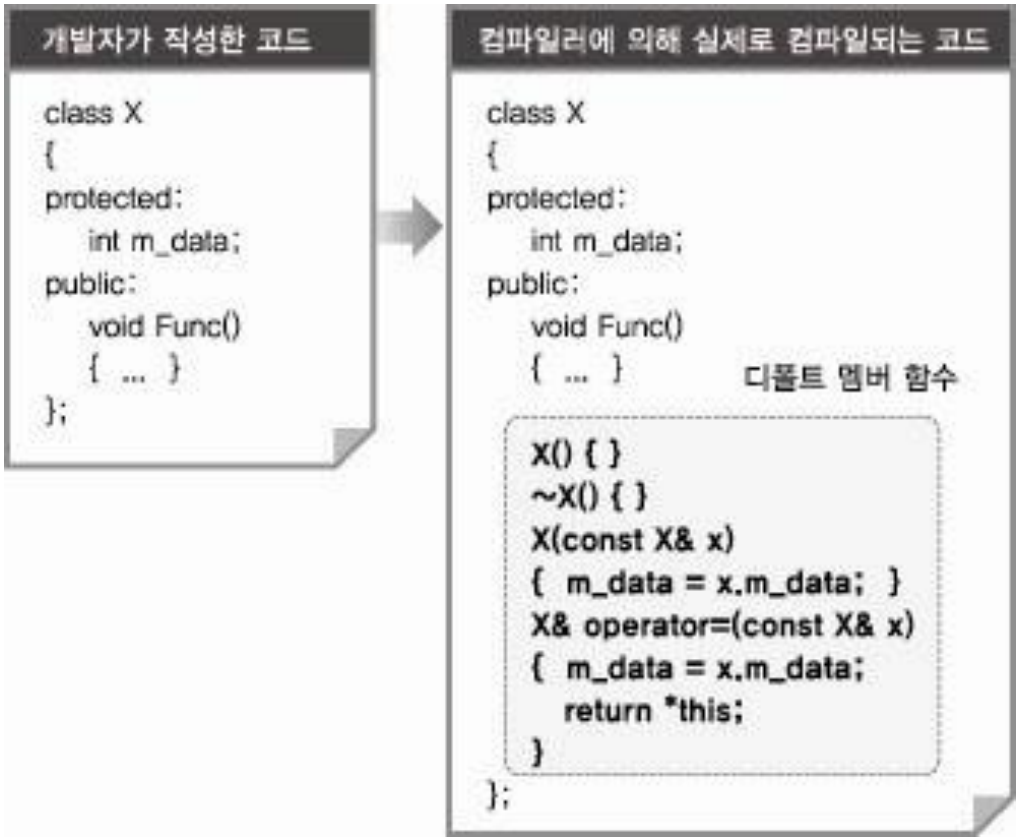
- 객체 간의 초기화 : **복사 생성자** 이용
  - 같은 클래스의 다른 객체와 같은 값을 갖도록 초기화
  - 클래스의 멤버 변수를 1대1로 초기화

- 객체 간의 대입 : **대입 연산자** 이용
  - 같은 클래스의 다른 객체의 값을 대입
  - 클래스의 멤버 변수를 1대1로 대입

```
Stack s1(5);
Stack s2 = s1;      // 객체 간의 초기화 = 복사생성자
Stack s3;
s3 = s1;            // 객체 간의 대입
```

디폴트 함수들

- 개발자가 따로 정의하지 않으면 컴파일러에 의해서 제공되는 함수
  - 디폴트 생성자
  - 디폴트 소멸자
  - 디폴트 복사 생성자
  - 디폴트 대입 연산자



5.2 분할 컴파일

바람직한 C++ 프로그램 작성법

- 클래스를 헤더 파일과 cpp 파일로 분리하여 작성
  - 클래스마다 분리 저장
  - 클래스 선언 부 : 헤더 파일(.h)에 저장
    → "**클래스 이름.h**"
  - 클래스 구현 부 : cpp 파일에 저장 (멤버함수의 구현)
    - 클래스가 선언된 헤더 파일 include

    → "**클래스 이름.cpp**"
  - main() 등 전역 함수나 변수는 다른 cpp 파일에 분산 저장
    - 필요하면 클래스가 선언된 헤더 파일 include
- 목적
  - 클래스 재사용

분할 컴파일

```
class Circle {
private:
    int radius;
public:
     void SetRadis();
     int GetRadius();
    double getArea();
};
```

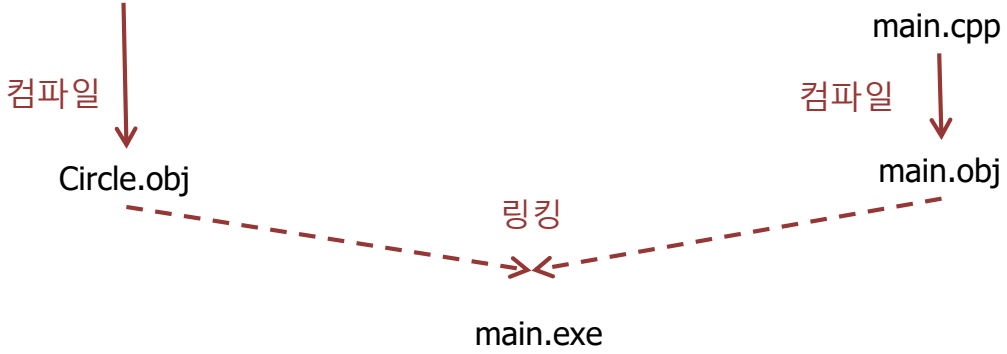
Circle.h

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

#include "Circle.h"

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

int getRadius() {
    return radius;
}

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

Circle.cpp

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

#include "Circle.h"

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

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

main.cpp

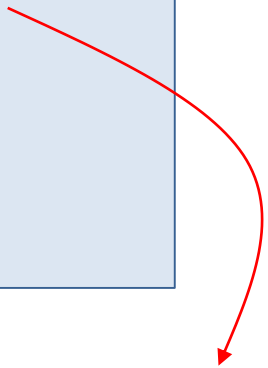



```
반지름 1 원 생성
donut 면적은 3.14
반지름 30 원 생성
pizza 면적은 2826
```

헤더 파일의 중복 include 문제

- 헤더 파일을 중복 include 할 때 생기는 문제

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

#include "Circle.h"
#include "Circle.h" // 컴파일 오류 발생
#include "Circle.h"

int main() {
        ..........
}
```



circle.h(4): error C2011: 'Circle' : 'class' 형식 재정의

헤더 파일의 중복 include 문제를 조건 컴파일로 해결

- 헤더의 중복 포함을 막는 방법
  - 규칙 1. 헤더 파일의 이름을 따서 심볼을 만든다 (예: TEST_H)
  - 규칙 2. 헤더 파일의 제일 앞에 이 심볼을 사용해서 #ifndef, #define 명령을 추가한다.
  - 규칙 3. 헤더 파일의 제일 끝에 #endif를 추가한다.

조건 컴파일 문의 상수(CIRCLE_H)는 다른 조건 컴파일 상수와 충돌을 피하기 위해 클래스의 이름으로 하는 것이 좋음.

조건 컴파일 문. Circle.h를 여러 번 include해도 문제 없게 하기 위함

```
#ifndef CIRCLE_H
#define CIRCLE_H

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

#endif
```

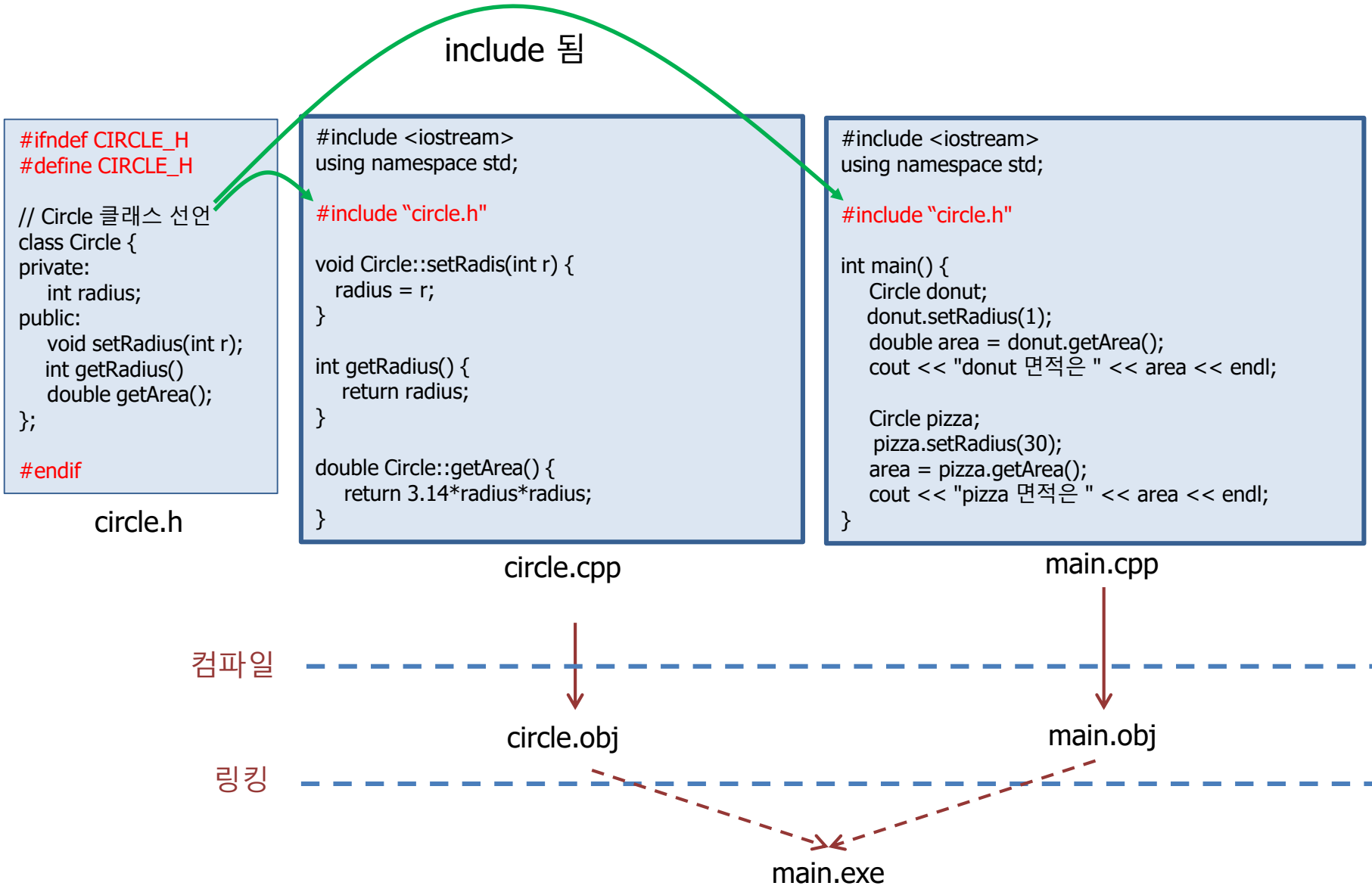
Circle.h

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

#include "Circle.h"
#include "Circle.h"
#include "Circle.h"

int main() {
   ..........
}
```

main.cpp

컴파일 오류 없음

헤더 파일의 중복 include 문제를 조건 컴파일로 해결



다음 수업

- 접근지정자, const객체, static멤버
  - 1_ 접근지정자와 접근자함수
  - 2_ const 객체와 const 멤버함수
  - 3_ static 멤버