04

객체 포인터와 객체 배열, 객체의 동적 생성

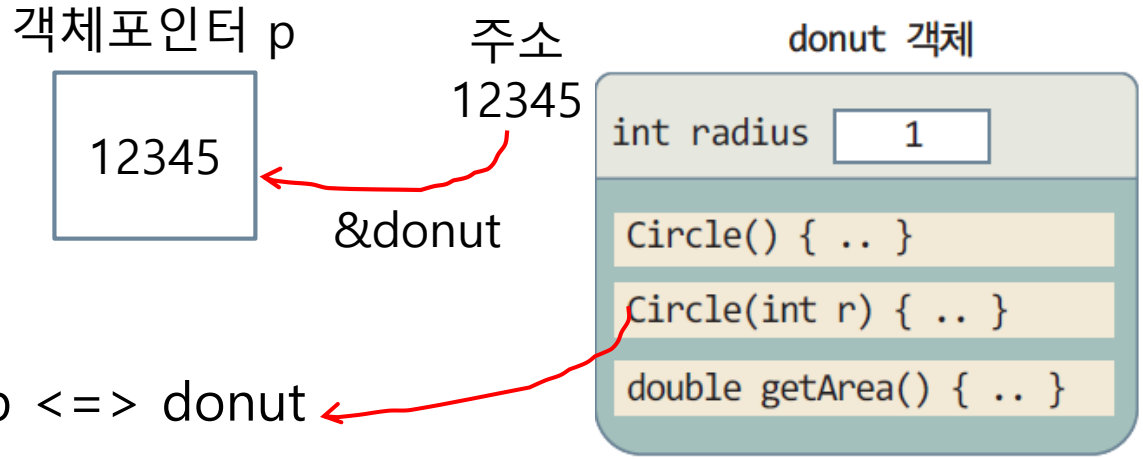
학습 목표

1. 객체에 대한 포인터를 선언하고 활용할 수 있다.
2. 객체의 배열을 선언하고 활용할 수 있다.
3. new를 이용하여 동적으로 메모리나 배열을 할당 받고 delete를 이용하여 반환할 수 있다.
4. new를 이용하여 동적으로 객체나 객체 배열을 할당 받고 delete를 이용하여 반환할 수 있다.
5. this 포인터의 개념을 이해하고, 활용할 수 있다.
6. string 클래스를 이용하여 문자열을 다룰 수 있다.

객체 포인터

- 객체의 주소를 저장하는 변수, 구조체 포인터와 유사
- 객체 포인터 선언
  - 클래스명* 객체포인터명;
- 객체와 객체 포인터의 차이
  - 객체는 멤버변수와 멤버함수가 저장된 변수
  - 객체포인터는 객체의 주소가 저장된 변수 -> 객체가 아님

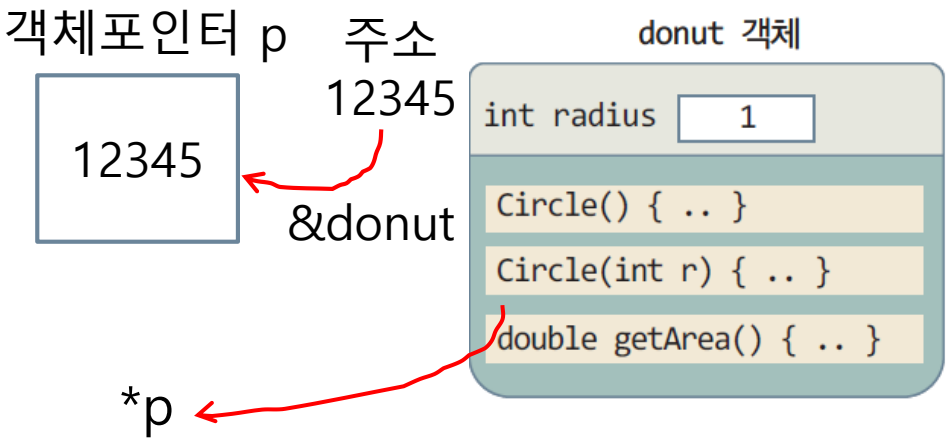
```
Circle donut;
Circle* p;
p = &donut;
```



객체 포인터

- 객체 포인터로 멤버를 접근
  - 객체포인터->멤버 // 구조체 포인터와 동일

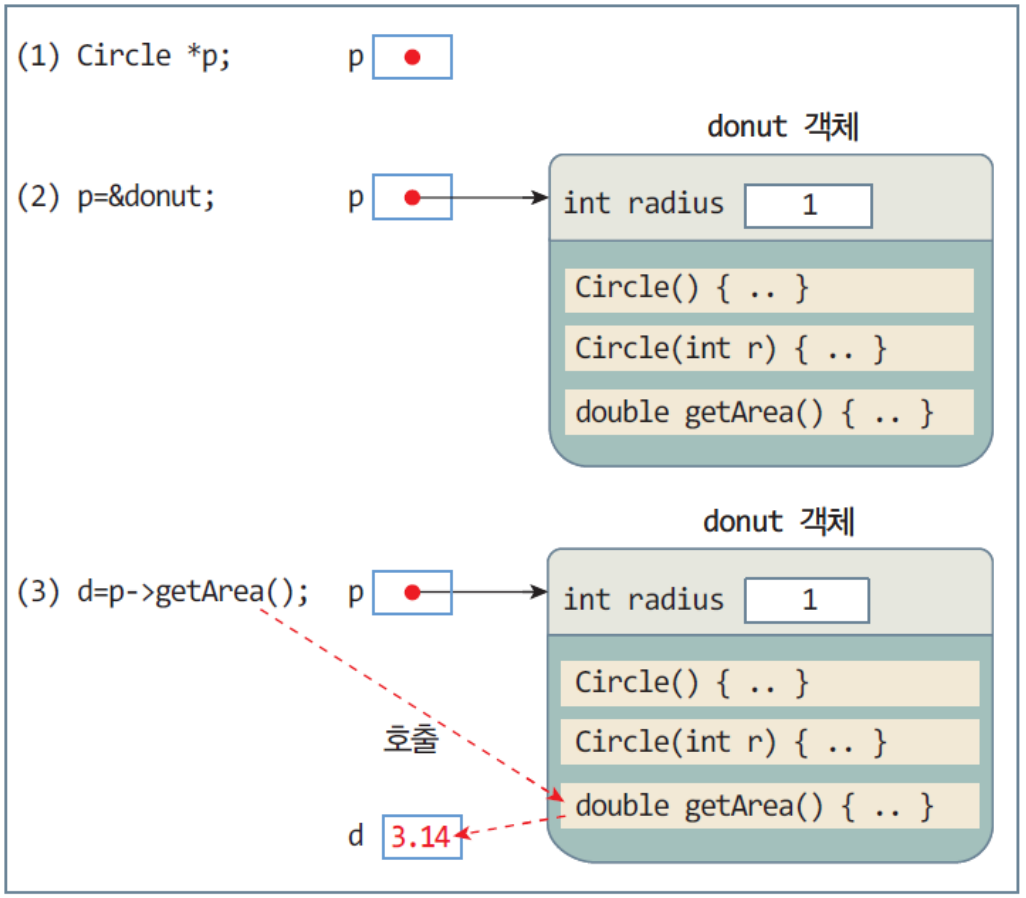
```
Circle donut;
double d = donut.getArea();

Circle* p;
p = &donut;
d = p->getArea();
```



(*p).radius => p->radius
(*p).getArea() => p->getArea()

객체 포인터


```
Circle donut;
double d = donut.getArea();

Circle* p;                  // (1)
p = &donut;                 // (2)
d = p->getArea();           // (3)
```



예제 4–1 객체 포인터 선언 및 활용


```
#include <iostream>
using namespace std;
class Circle {
	int radius;
public:
	Circle() { radius = 1; }
	Circle(int r) { radius = r; }
	double getArea();
};

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

```
3.14
3.14
3.14
2826
2826
```

예제 4–1 객체 포인터 선언 및 활용


```
int main() {
        Circle donut;
        Circle pizza(30);
        cout << donut.getArea() << endl;        // 객체 이름으로 멤버 접근

        Circle* p;                              // 객체 포인터로 멤버 접근
        p = &donut;
        cout << p->getArea() << endl;           // donut의 getArea() 호출
        cout << (*p).getArea() << endl;         // *p는 donut과 같음

        p = &pizza;
        cout << p->getArea() << endl;           // pizza의 getArea() 호출
        cout << (*p).getArea() << endl;         // *p는 pizza와 같음

}
```

```
3.14
3.14
3.14
2826
2826
```

this 포인터

- 객체 자신을 가리키는 객체포인터
- 컴파일러에 의해 모든 멤버함수에 자동으로 삽입되는 매개변수
  - 개발자가 선언하는 변수가 아니고, 컴파일러가 선언한 변수
- 클래스의 멤버 함수 내에서만 사용 가능하고 멤버 함수가 아닌 함수에서 this 사용 불가

```
class Circle {
    int radius;
public:
    Circle() { this->radius = 1; }
    Circle(int radius) { this->radius = radius; }
    void setRadius(int radius) { this->radius = radius; }
    ....
};
```

this 포인터의 실체 – 컴파일러에서 처리


```
class Sample {
      int a;
public:
      void setA(int x) {
            this->a = x;
      }
};
...
Sample ob;
ob.setA(5);
```

컴파일러에 의해 자동변환

```
class Sample {
      int a;
public:
      void setA(Sample* this, int x) {
            this->a = x;
      }
};
...
Sample ob;
ob.setA(&ob, 5);
```

this는 컴파일러에 의해 묵시적으로 삽입된 매개 변수

ob의 주소가 this 매개변수에 전달됨

(a) 개발자가 작성한 코드

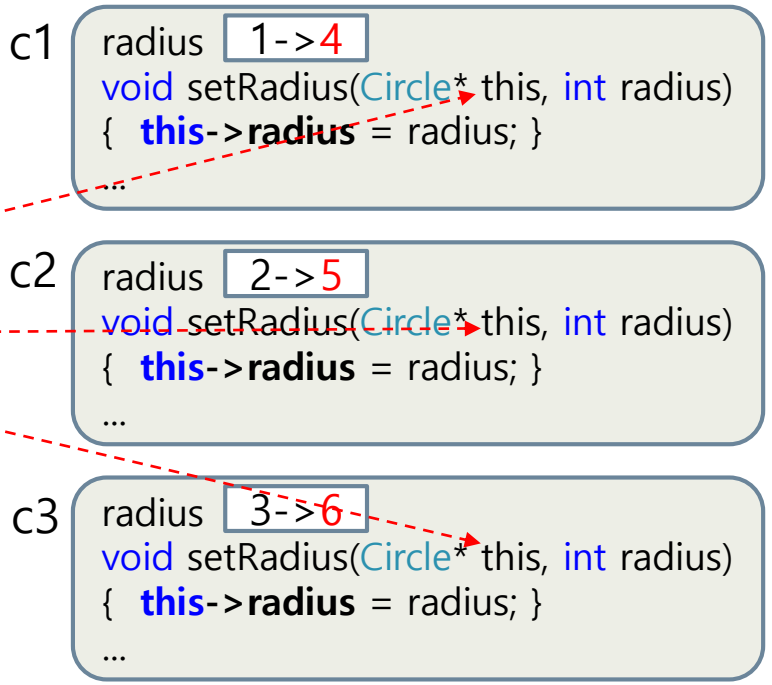
(b) 컴파일러에 의해 변환된 코드

this와 객체


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

class Circle {
	int radius;
public:
	Circle() { this->radius = 1; }
	Circle(int radius) { this->radius = radius; }
	void setRadius(int radius){ this->radius = radius; }
};
```

this와 객체


```
int main() {
        Circle c1;
        Circle c2(2);
        Circle c3(3);
        c1.setRadius(4); //c1.setRadius(&c1, 4);
        c2.setRadius(5); //c2.setRadius(&c2, 5);
        c3.setRadius(6); //c3.setRadius(&c3, 6);
        return 0;
}
```



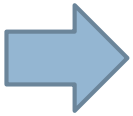
this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우

```
Circle(int radius) {
    radius = radius;
}
```

매개 변수 radius

매개 변수 radius



```
Circle(int radius) {
    this->radius = radius;
}
```

멤버 radius

매개 변수 radius

- 멤버 함수가 객체 자신의 주소를 리턴할 때
  - 연산자 중복 시에 매우 필요

```
class Sample {
public:
    Sample* f() {
        ....
        return this;
    }
};
```

객체 배열, 생성 및 소멸

- 객체 배열 선언 : 기본 자료형의 배열 선언과 동일
  - int n[3]; // 정수형 배열 선언
  - Circle c[3]; // Circle 타입의 배열 선언
- 객체 배열 생성 과정
  1. 객체 배열을 위한 공간 할당
  2. 배열의 각 원소 객체마다 생성자 실행
     - c[0]의 생성자 -> c[1]의 생성자 -> c[2]의 생성자 실행
     - 매개 변수 없는 디폴트 생성자 호출
- 객체 배열 소멸 과정
  - 배열의 각 원소객체마다 소멸자 호출. 생성의 반대순으로 소멸
    - c[2]의 소멸자 -> c[1]의 소멸자 -> c[0]의 소멸자 실행

예제 4– 2 Circle 클래스의 배열 선언 및 활용


```
#include <iostream>
using namespace std;
class Circle {
	int radius;
public:
	Circle() { radius = 1; }
	Circle(int r) { radius = r; }
	void setRadius(int r) { radius = r; }
	double getArea();
};
double Circle::getArea() {
	return 3.14 * radius * radius;
}
```

예제 4– 2 Circle 클래스의 배열 선언 및 활용

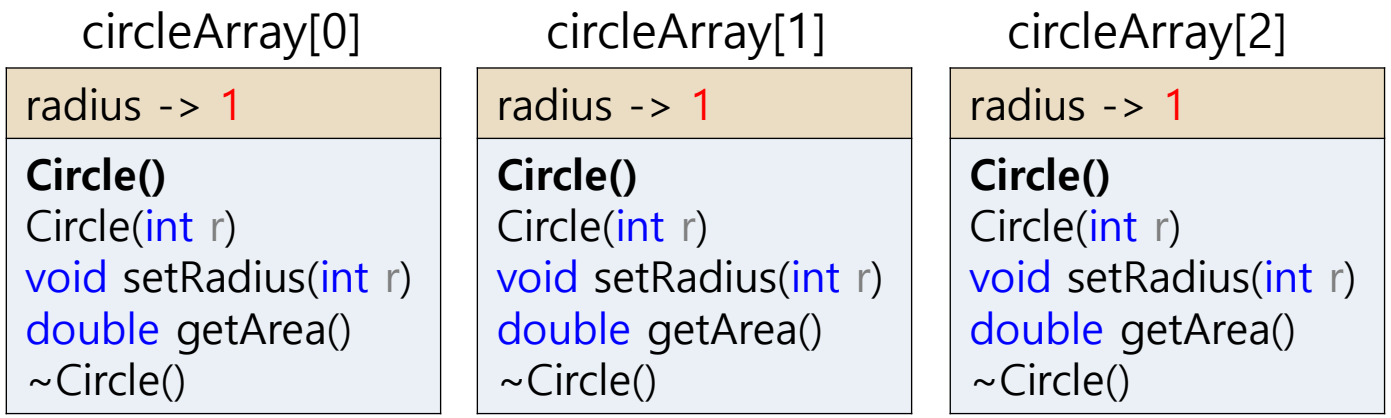

```
int main() {
	Circle circleArray[3];
	circleArray[0].setRadius(10);
	circleArray[1].setRadius(20);
	circleArray[2].setRadius(30);

	for (int i = 0; i < 3; i++)
		cout <<"Circle"<< i<<"의 면적은"<< circleArray[i].getArea() << endl;

	return 0;
}
```

```
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 2826
```

배열 생성과 활용(예제 4-2의 실행 과정)


```
Circle circleArray[3];
```

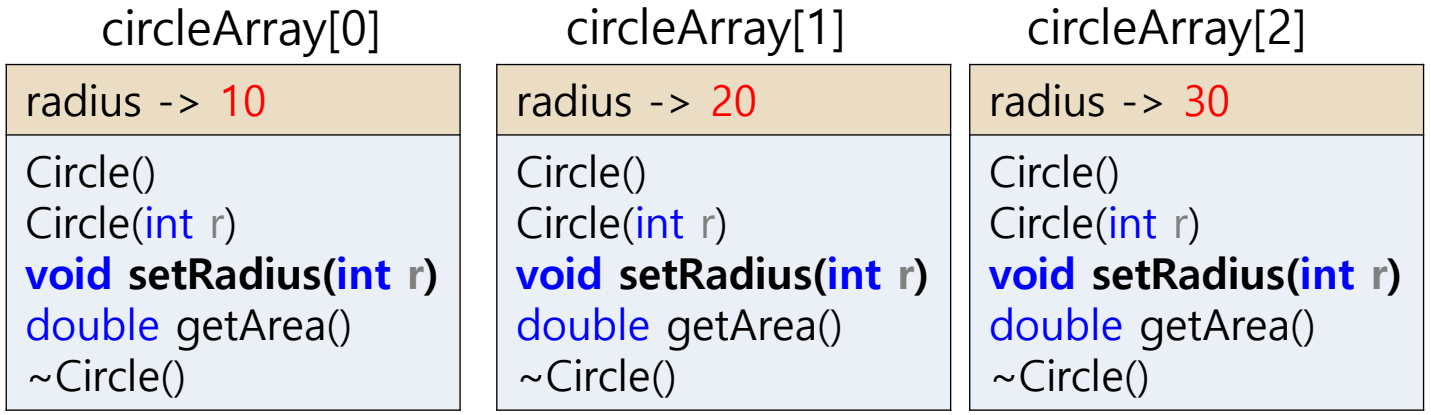
- 객체배열이 메모리에 생성된 후 각 원소별로 생성자 호출됨
- circleArray[0].Circle() ->circleArray[1].Circle() -> circleArray[2].Circle()



배열 생성과 활용(예제 4-2의 실행 과정)


```
circleArray[0].setRadius(10);
circleArray[1].setRadius(20);
circleArray[2].setRadius(30);
```

- 객체 배열원소의 멤버 함수 호출



배열 생성과 활용(예제 4-2의 실행 과정)

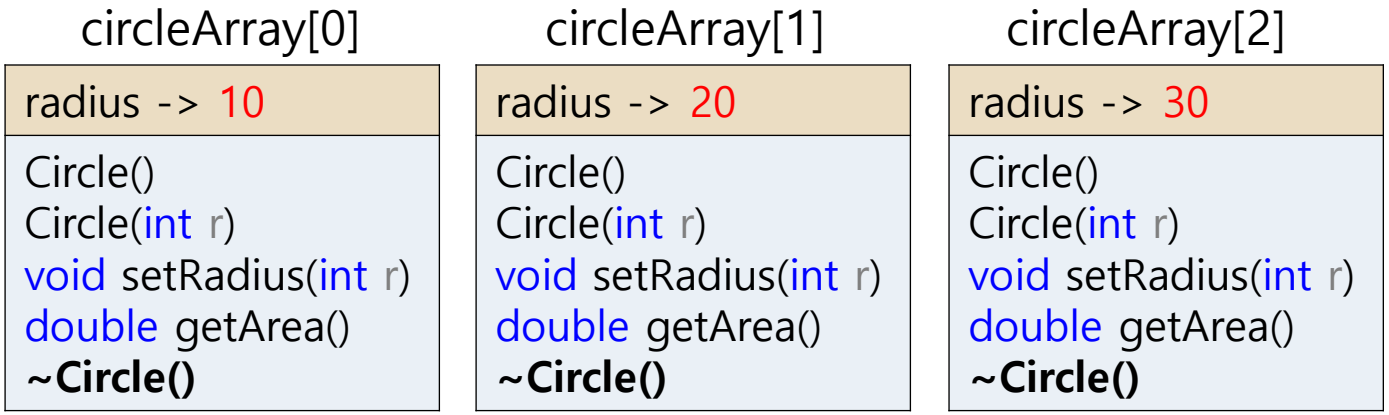

```
circleArray[i].getArea()
```

- 객체 배열원소의 멤버 함수 호출

circleArray[0]	circleArray[1]	circleArray[2]
radius -> 10	radius -> 20	radius -> 30
Circle() Circle(int r) void setRadius(int r) **double getArea()** ~Circle()	Circle() Circle(int r) void setRadius(int r) **double getArea()** ~Circle()	Circle() Circle(int r) void setRadius(int r) **double getArea()** ~Circle()

배열 생성과 활용(예제 4-2의 실행 과정)

- 객체 배열 소멸
  - 함수가 종료되면 함수내에서 선언된 객체배열도 소멸
  - 생성된 반대 순서대로 소멸자 자동 호출됨
  - circleArray[2].~Circle()->circleArray[1].~Circle()->circleArray[0].~Circle()



예제 4– 2 객체 포인터를 이용

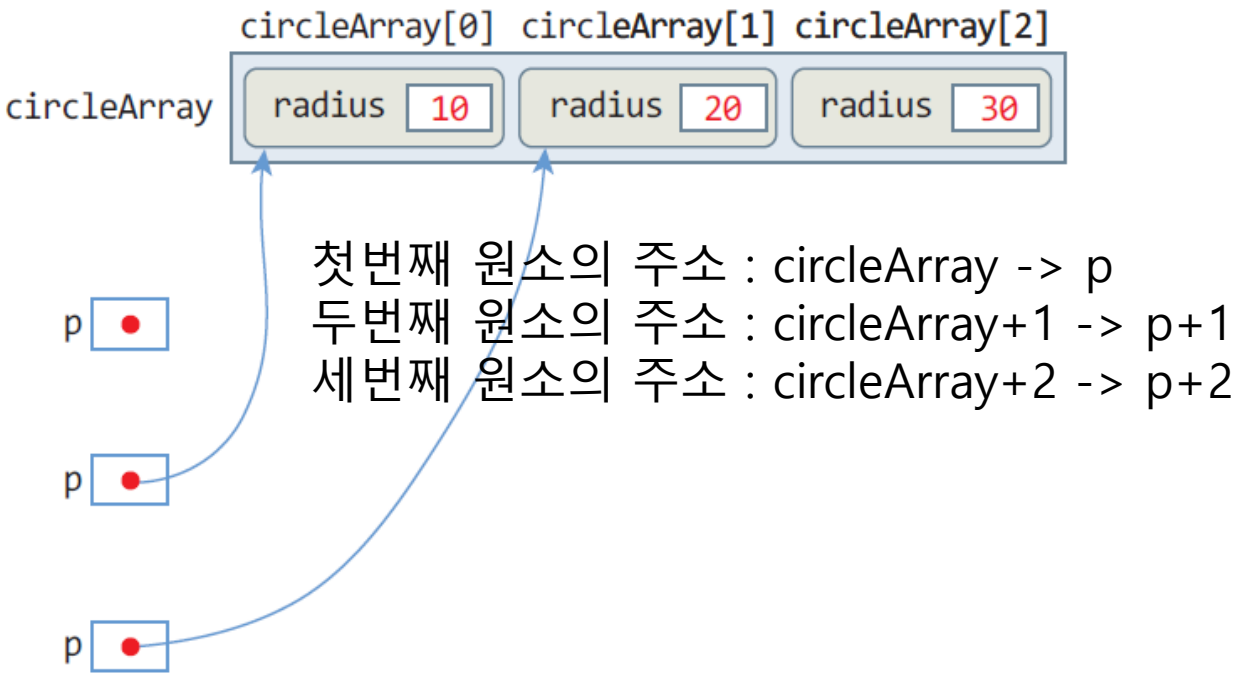

```
int main() {
        Circle circleArray[3];
        Circle* p;
        p = circleArray;
        p->setRadius(10);                   // p[0].setRadius(10);
        (p+1)->setRadius(20);               // p[1].setRadius(20);
        (p+2)->setRadius(30);               // p[2].setRadius(30);

        for (int i = 0; i < 3; i++) {
                cout << "Circle " << i << "의 면적은 " << p->getArea() << endl;
                p++;
        }
        return 0;
}
```

```
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 2826
```

배열 생성과 활용(예제 4-2의 실행 과정)

- 객체 포인터 활용

```
(3) Circle *p;

(4) p = circleArray;

(5) p++;
```



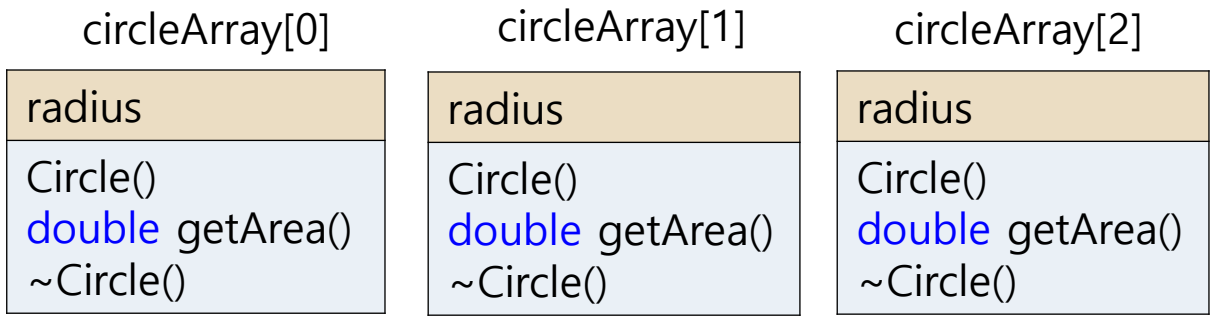
다양한 객체포인터 활용법


```
int main() {
	Circle circleArray[3];
	Circle* p = circleArray;

	for (int i = 0; i < 3; i++)
		cout << (circleArray+i)->getArea() << endl;

	for (int i = 0; i < 3; i++)
		cout << (p+i)->getArea() << endl;

	for (int i = 0; i < 3; i++)
		cout << p[i].getArea() << endl;

	for (int i = 0; i < 3; i++) {
		cout << p->getArea() << endl;
		p++;
	}
	return 0;
}
```

객체 배열 생성시 디폴트 생성자 호출


```
#include <iostream>
using namespace std;
class Circle {
	int radius;
public:
	double getArea() { return 3.14 * radius * radius; }
};
int main() {
	Circle circleArray[3];    // 디폴트 생성자 Circle() 호출,
	return 0;
}
```

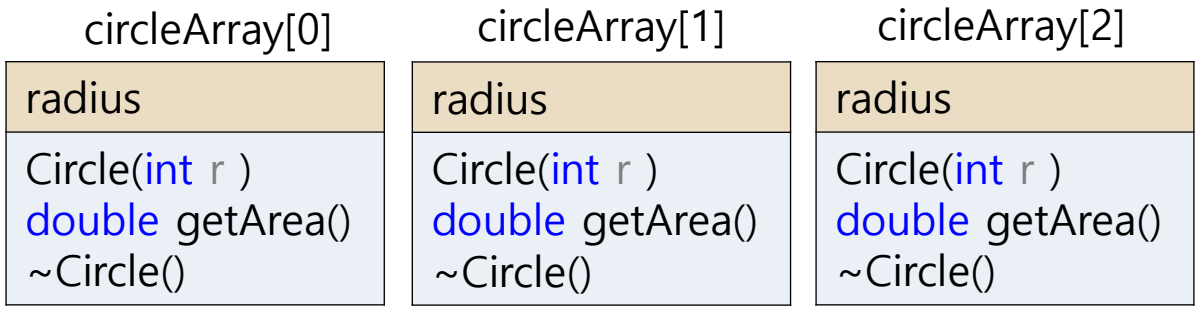
(a) 생성자가 선언되어 있지 않은 Circle 클래스



객체 배열 생성시 디폴트 생성자 호출


```
#include <iostream>
using namespace std;
class Circle {
	int radius;
public:
	Circle(int r) { radius = r; }
	double getArea() { return 3.14 * radius * radius; }
};
int main() {
	Circle waffle(15);		// Circle(int r) 호출
	Circle circleArray[3];	// 디폴트 생성자 Circle() 호출 ->컴파일 오류
	return 0;
}
```

(b) 디폴트 생성자가 없으므로 컴파일 오류



객체 배열 초기화

- 중괄호안에 배열의 각 원소별로 호출할 생성자를 지정

```
Circle circleArray[3] = { Circle(10), Circle(20), Circle() };
Circle circleArray[3] = { Circle(10), Circle(20) };
Circle circleArray[3] = {10, 20, 30 }; //{Circle(10),Circle(20),Circle(30)};
Circle circleArray[3] = {10, 20 };       //{Circle(10),Circle(20),Circle()};
```

  - circleArray[0] 객체가 생성될 때, 생성자 Circle(10) 호출
  - circleArray[1] 객체가 생성될 때, 생성자 Circle(20) 호출
  - circleArray[2] 객체가 생성될 때, 생성자 Circle() 호출
  - 없으면 매개변수 없는 생성자 호출
  - 상수이면 상수를 받을 수 있는 생성자 호출
- 잘못된 초기화 방법

```
Circle circleArray[3](3);                    // 오류
```

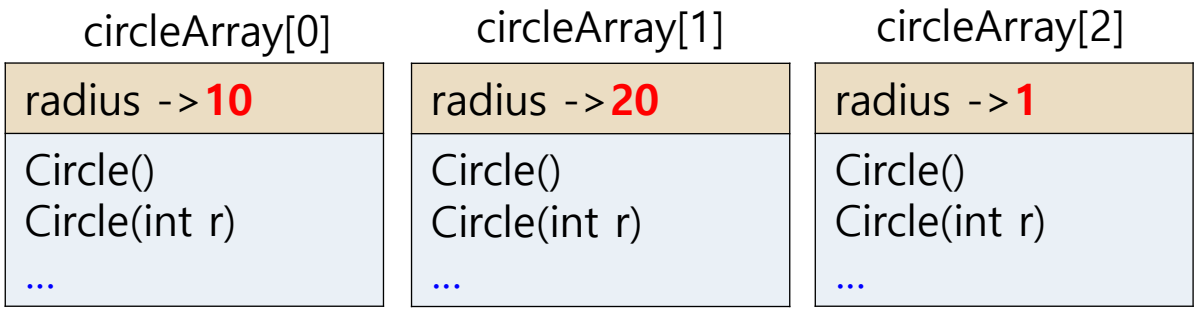
예제 4–3 객체 배열 초기화


```
#include <iostream>
using namespace std;
class Circle {
	int radius;
public:
	Circle() { radius = 1; }
	Circle(int r) { radius = r; }
	void setRadius(int r) { radius = r; }
	double getArea();
};
double Circle::getArea() { return 3.14 * radius * radius; }
```

```
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 3.14
```

예제 4–3 객체 배열 초기화


```
int main() {
	Circle circleArray[3] = { Circle(10), Circle(20), Circle() };
	for (int i = 0; i < 3; i++)
		cout <<"Circle "<< i <<"의 면적은 "<< circleArray[i].getArea()
		<< endl;
	return 0;
}
```



다차원 객체 배열

- 생략, 각자 공부해볼 것 -> C언어 교재에서 2차원 배열을 먼저 공부해야 함

실습과제1

- 배열명의 의미를 설명하라.
- 객체 포인터의 의미를 설명하라.
- 배열과 포인터사이의 관계를 설명하라.

실습과제2

- 지난과제에서 다룬 삼각형 클래스 Triangle 를 만들고 아래 멤버함수를 호출하는 부분을 객체포인터를 이용하여 호출하는 코드로 변경 하시오.

```
// 클래스 정의 추가
int main() {
    Triangle tri;
    // 객체포인터변수 선언 및 초기화 코드 추가
    tri.setWidth(3);          // 객체포인터 표현으로 변경
    tri.setHeight(5);         // 객체포인터 표현으로 변경
    cout <<"삼각형의 면적은"<< tri.getArea()<< endl;  //포인터표현으로 변경
    return 0;
}
```

삼각형의 면적은 7.5

실습과제3

- 삼각형 클래스 Triangle 를 만들고 모든 멤버함수를 this포인터를 이용하여 구현하라.

```
// 클래스 정의 추가
int main() {
    Triangle tri1;          // 밑변=높이=1로 초기화
    cout << "삼각형의 면적은 " << tri1.getArea() << endl;
    Triangle tri2(10);      // 밑변=10,높이=1로 초기화
    cout << "삼각형의 면적은 " << tri2.getArea() << endl;
    Triangle tri3(10,2);    // 밑변=10,높이=2로 초기화
    cout << "삼각형의 면적은 " << tri3.getArea() << endl;
    return 0;
}
```

삼각형의 면적은 0.5
삼각형의 면적은 5
삼각형의 면적은 10

실습과제4

- 삼각형 클래스 Triangle를 만들고 객체 배열 3개를 선언 후 각각 밑변=높이=2,4,6으로 초기화하고 면적을 출력하는 코드로 수정하라.

```
삼각형0의 면적은 2
삼각형1의 면적은 8
삼각형2의 면적은 18
```

실습과제5

- 실습과제3번을 객체 포인터를 이용한 코드로 변경해보라

```
삼각형0의 면적은 2
삼각형1의 면적은 8
삼각형2의 면적은 18
```

과제제출방법

- 소스코드, 라인단위의 주석, 실행결과를 포함하는 pdf파일을 작성한 후 eclass 과제 게시판에 업로드, **반드시 하나의 pdf파일로 업로드할 것**
- 기한 : 과제 게시판에 마감시간 참조
- 실행결과를 캡쳐할 때 글자를 알아보기 쉽게 확대해서 캡쳐할 것.
- 소스코드의 첫 부분은 아래처럼 제목,날짜,작성자(학번,이름)를 작성할 것

```
// ************************************************
//    제   목 :  정수 4개의 평균을 구하는 프로그램
//    날   짜 :  2023년 9월10일
//    작성자 :  15010101 홍길동
// ************************************************

// 소스코드 작성
```