# 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



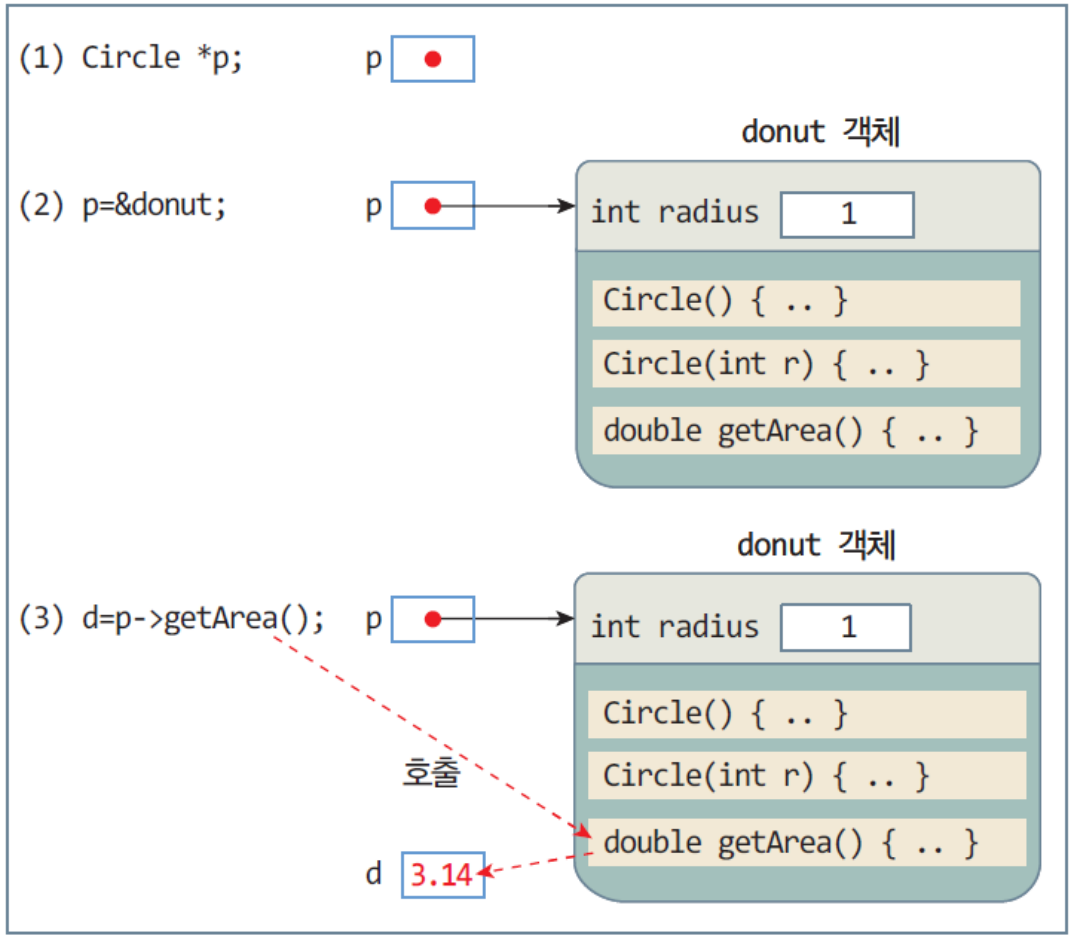
- 객체에 대한 포인터
  - C 언어의 포인터와 동일
  - 객체의 주소 값을 가지는 변수
- 포인터로 멤버를 접근할 때
  - 객체포인터->멤버

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

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

객체에 대한 포인터 선언

포인터에 객체 주소 저장

멤버 함수 호출



# 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;
}
```

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

   // 객체 이름으로 멤버 접근
   cout << donut.getArea() << endl;

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

   p = &pizza;
   cout << p->getArea() << endl; // pizza의 getArea() 호출
   cout << (*p).getArea() << endl; // pizza의 getArea() 호출
}
```

```
3.14
3.14
3.14
2826
2826
```

# 객체 배열, 생성 및 소멸



- 객체 배열 선언 가능
  - 기본 타입 배열 선언과 형식 동일
    - int n[3]; // 정수형 배열 선언
    - Circle c[3]; // Circle 타입의 배열 선언
- 객체 배열 선언
  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 생성자 호출
  - 매개 변수 있는 생성자를 호출할 수 없음
    - Circle circleArray[3](5); // 오류
- 배열 소멸
  - 배열의 각 객체마다 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;
}
```

```
int main() {
    Circle circleArray[3];                          // (1) Circle 객체 배열 생성

    // 배열의 각 원소 객체의 멤버 접근
    circleArray[0].setRadius(10);                   // (2)
    circleArray[1].setRadius(20);
    circleArray[2].setRadius(30);

    for(int i=0; i<3; i++) // 배열의 각 원소 객체의 멤버 접근
        cout << "Circle " << i << "의 면적은 " << circleArray[i].getArea() << endl;

    Circle *p;                                      // (3)
    p = circleArray;                                // (4)
    for(int i=0; i<3; i++) {  // 객체 포인터로 배열 접근
        cout << "Circle " << i << "의 면적은 " << p->getArea() << endl;
        p++;                                        // (5)
    }
}
```

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

# 배열 생성과 활용(예제 4-2의 실행 과정)



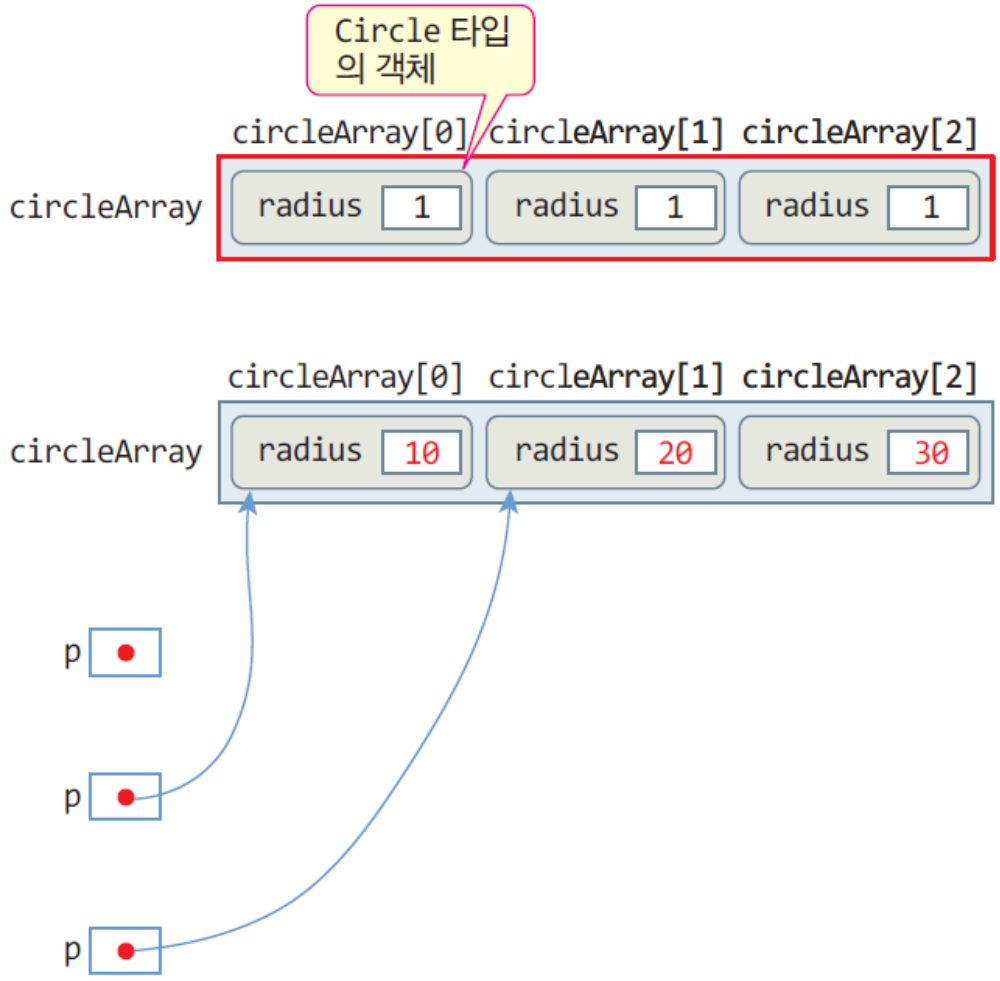
(1) Circle circleArray[3];

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

(3) Circle *p;

(4) p = circleArray;

(5) p++;



# 객체 배열 생성시 기본 생성자 호출



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

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

int main() {
   Circle circleArray[3];
}
```

컴파일러가 자동으로 기본 생성자 Circle() { } 삽입.
컴파일 오류가 발생하지 않음

기본 생성자 Circle() 호출

(a) 생성자가 선언되어 있지 않은 Circle 클래스

```
#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 circleArray[3];
}
```

Circle(int r) 호출

기본 생성자 Circle() 호출. 기본 생성자가 없으므로 컴파일 오류

error.cpp(15): error C2512: 'Circle' : 사용할 수 있는 적절한 기본 생성자가 없습니다

(b) 기본 생성자가 없으므로 컴파일 오류

# 객체 배열 초기화



- 객체 배열 초기화 방법
  - 배열의 각 원소 객체당 생성자 지정하는 방법

```
Circle circleArray[3] = { 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

# 예제 4–3 객체 배열 초기화



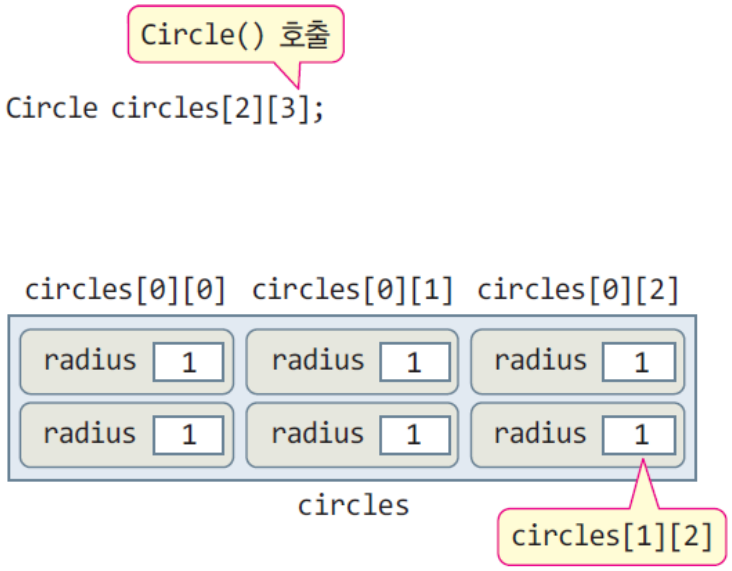
```
#include <iostream>
using namespace std;

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;
}

int main() {
    Circle circleArray[3] = { Circle(10), Circle(20), Circle() }; // Circle 배열 초기화

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

circleArray[0] 객체가 생성될 때, 생성자 Circle(10),
circleArray[1] 객체가 생성될 때, 생성자 Circle(20),
circleArray[2] 객체가 생성될 때, 기본 생성자 Circle()
이 호출된다.

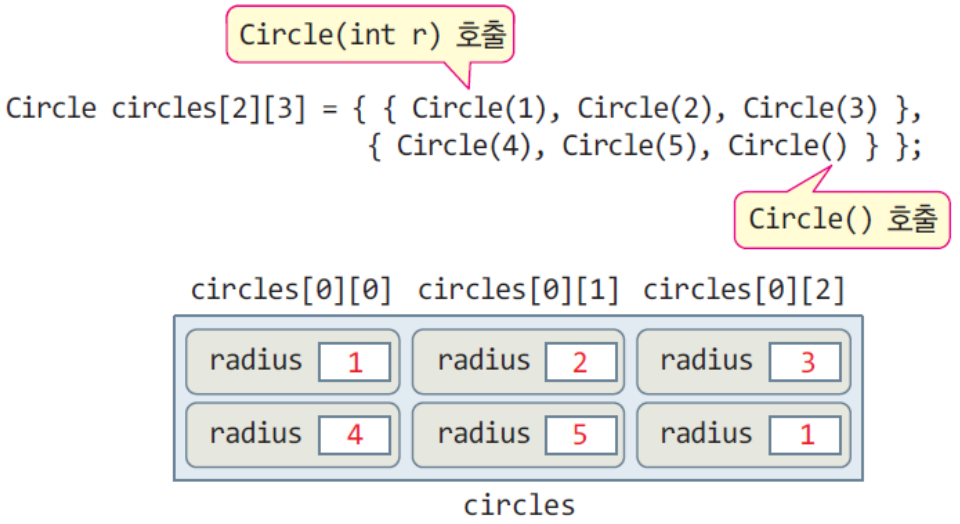
```
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 3.14
```

# 2차원 배열






(a) 2차원 배열 선언 시




(b) 2차원 배열 선언과 초기화

```
circles[0][0].setRadius(1);
circles[0][1].setRadius(2);
circles[0][2].setRadius(3);
circles[1][0].setRadius(4);
circles[1][1].setRadius(5);
circles[1][2].setRadius(6);
```

2차원 배열을 초기화하는 다른 방식

# 예제 4-4 Circle 클래스의 2차원 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;
}
```

```
int main() {
    Circle circles[2][3];

    circles[0][0].setRadius(1);
    circles[0][1].setRadius(2);
    circles[0][2].setRadius(3);
    circles[1][0].setRadius(4);
    circles[1][1].setRadius(5);
    circles[1][2].setRadius(6);

    for(int i=0; i<2; i++)    // 배열의 각 원소 객체의 멤버 접근
        for(int j=0; j<3; j++) {
            cout << "Circle [" << i << "," << j  << "]의 면적은 ";
            cout << circles[i][j].getArea() <<endl;
        }
}
```

```
Circle circles[2][3] =
    { { Circle(1), Circle(2), Circle(3) },
      { Circle(4), Circle(5), Circle() } };
```

```
Circle [0,0]의 면적은 3.14
Circle [0,1]의 면적은 12.56
Circle [0,2]의 면적은 28.26
Circle [1,0]의 면적은 50.24
Circle [1,1]의 면적은 78.5
Circle [1,2]의 면적은 113.04
```