5장 생성자와 접근제어

2020. 9. 21

순천향대학교 컴퓨터 공학과

내용

- 생성자
- 소멸자
- 접근자와 설정자
- 객체와 함수

생성자

- 생성자(constructor)는 객체 초기화를 담당하는 함수

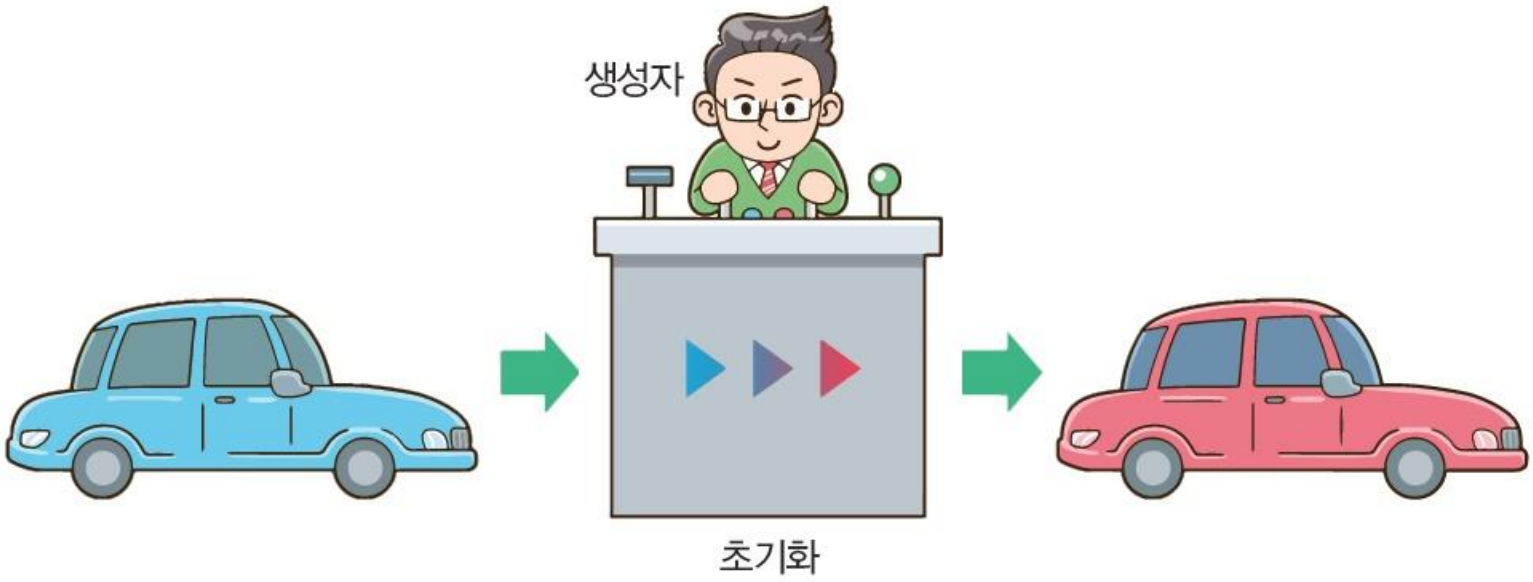


그림 5.1 생성자의 역할

생성자

- 생성자(constructor)
  - 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
  - 생성자 이름은 클래스 이름과 동일
  - 반환 값 타입이 없음

```
class Circle {
        ..............
        Circle();
        Circle(int r);
        ..........................................
};

Circle::Circle() {
        ..............
}

Circle::Circle(int r) {
        ..............
}
```

클래스 이름과 동일

리턴 타입 명기하지 않음

생성자의 특징(1)

- 생성자 역할
  - 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. (사용자에 의해서)
  - 각 객체마다 생성자 실행

생성자의 특징(2)

- 생성자는 중복 가능
  - 생성자는 한 클래스 내에 여러 개 존재 가능
  - 중복된 생성자 중 하나만 실행

- 생성자가 선언되어 있지 않으면 기본 생성자가 자동으로 생성
  - 기본 생성자는 매개 변수가 없는 생성자를 말함
  - 컴파일러에 의해 자동 생성

예제: 생성자를 가진 Circle 클래스

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


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

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

객체 생성 및 생성자 실행 과정

Circle 클래스

```
int radius
Circle() {
    radius = 1;
}
Circle(int r) {
    radius = r;
}
double getArea() {
    ....
}
```

Circle donut;

❶ 객체 생성(객체 공간 할당)

Circle pizza(30);

donut 객체

```
int radius
Circle() {
    radius = 1;
}
Circle(int r) {
    radius = r;
}
double getArea() {
    ....
}
```

pizza 객체

```
int radius
Circle() {
    radius = 1;
}
Circle(int r) {
    radius = r;
}
double getArea() {
    ....
}
```

❷ 생성자 실행

```
int radius 1
Circle() {
    radius = 1;
}
Circle(int r) {
    radius = r;
}
double getArea() {
    ....
}
```

```
int radius 30
Circle() {
    radius = 1;
}
Circle(int r) {
    radius = r;
}
double getArea() {
    ....
}
```

기본 생성자

1. 생성자는 꼭 있어야 하는가?
   - Yes!

2. 개발자가 클래스에 생성자를 작성해 놓지 않으면?
   - 컴파일러가 기본 생성자를 자동으로 생성

- 기본 생성자란?
  - 매개 변수 없는 생성자
  - 디폴트 생성자라고도 부름

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

기본 생성자의 자동 생성

- 생성자가 하나도 작성되어 있지 않은 경우 => 컴파일러가 기본 생성자를 자동 생성

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

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

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

Circle::Circle() {
}

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

컴파일러에 의해 자동으로 삽입됨

기본 생성자가 자동으로 생성되지 않는 경우

- 생성자가 하나라도 선언된 경우 => 컴파일러는 기본 생성자를 자동 생성하지 않음
- 다음 코드에 오류가 존재하는가?

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

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

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

멤버 초기화 리스트

- 생성자를 편리하게 작성하는 방법

```
Time(int h, int m) {
   hour = h;
   minute = m;
}
```

```
Time(int h, int m) : hour{h}, minute{m}
{ }
```

```
Time a;
Time b(10, 25);
Time c { 10, 25 };
Time d = { 10, 25 };
```

소멸자

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

```
class Circle {
    Circle();
    Circle(int r);
    ..............
    ~Circle();
};

Circle::~Circle() {
    ..............
}
```

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 없음

소멸자 특징 (2)

- 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
  - 기본 소멸자는 아무 일도 하지 않고 단순 리턴

```
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;
}

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

  return 0;
}
```

예제: 소멸자


```
#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;
}

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

   return 0;
}
```

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

객체는 생성의 역순으로 소멸된다.

소멸자 실행

- 객체가 언제 소멸되는가?
  - 함수 내에 선언된 객체는 함수 종료시에
  - 함수의 바깥에 선언된 객체는 프로그램 종료시에
- 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 (8장)
  - new를 실행하는 순간 객체 생성되고,
  - delete 연산자를 실행할 때 객체 소멸

예제: 소멸자

```
class MyString {
private:
        char *s;
        int size;
public:
        MyString(char *c) {
                size = strlen(c)+1;
                s = new char[size];
                strcpy(s, c);
        }
        ~MyString() {
                delete[] s;
        }
};
int main() {
        MyString str("abcdefghijk");
}
```

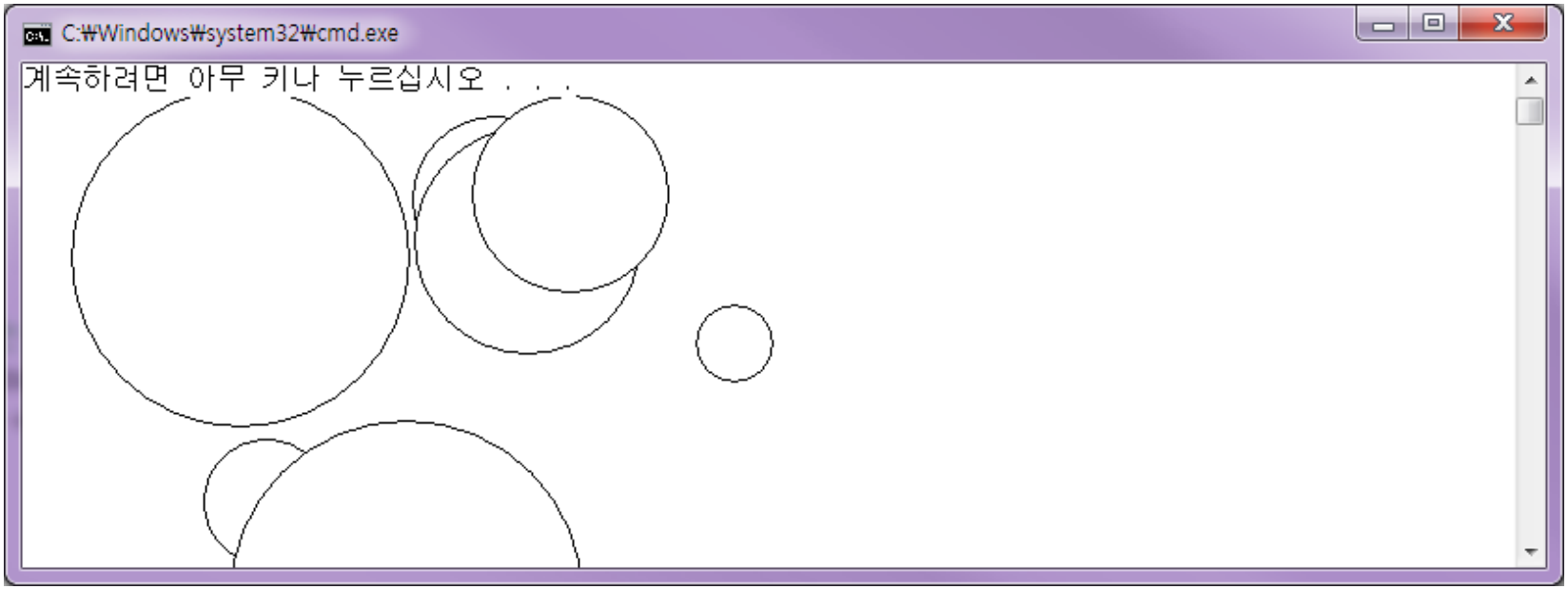
Lab: Rect 클래스

```
using namespace std;

class Rectangle {
        int width, height;
public:
        Rectangle(int w, int h);
        int calcArea();
};

Rectangle::Rectangle(int w, int h)
{
        width = w;
        height = h;
}

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

```
int main()
{
        Rectangle r{ 3, 4 };

        cout << "사각형의 넓이 : "
                << r.calcArea() << '\n';
        return 0;
}
```

Lab: Circle 클래스

- 원을 나타내는 **Circle** 클래스를 작성하여 보았다. 원은 중심점과 반지름과 색상으로 표현된다. 약 **10**개의 **Circle** 객체를 생성하면서 랜덤한 위치와 랜덤한 반지름으로 화면에 원을 그려보자. 원의 중심점과 반지름은 생성자를 호출하여 설정하라



```
#include <windows.h> // required for window based application

class Circle
{
        int x, y, radius;
        string color;
public:
        Circle(int xval = 0, int yval = 0, int r = 0, string c = "");
        double calcArea()       { return radius*radius*3.14;    }
        void draw();
};

Circle::Circle(int xval, int yval, int r, string c)
{
        x = xval;
        y = yval;
        radius = r;
        color = c;
}
```

```
int main() {
   for (int i=0; i<10; i++) {
        int x = 100+rand()%300;
        int y = 100 +rand() %300;
        int r = rand() %100;
        circle c {x, y, r, “yellow”};
        c.draw();
   }
   return 0;
}
```

- Window GDI(graphics device interface) 라이브러리 이용
  - 애플리케이션은 GDI를 통해서 원, 사각형, 직선 등의 그래픽스를 화면이나 프린터에 출력 가능

```
void circle::draw() {
   HDC hdc = GetWindowDC(GetForegroundWindow());
            // get a device context for the foreground window
            // which the user is currently working
            // device context permits painting anywhere in a window

   Ellipse(hdc, x-radius, y-radius, x+radius, y+radius);
            // 타원을 그린다
            // (hdc: device context,
            //  left, top,        : 사각형의 좌측 위 좌표
            //  right, bottom)  : 사각형의 우측 아래 좌표
}
```

접근 제어

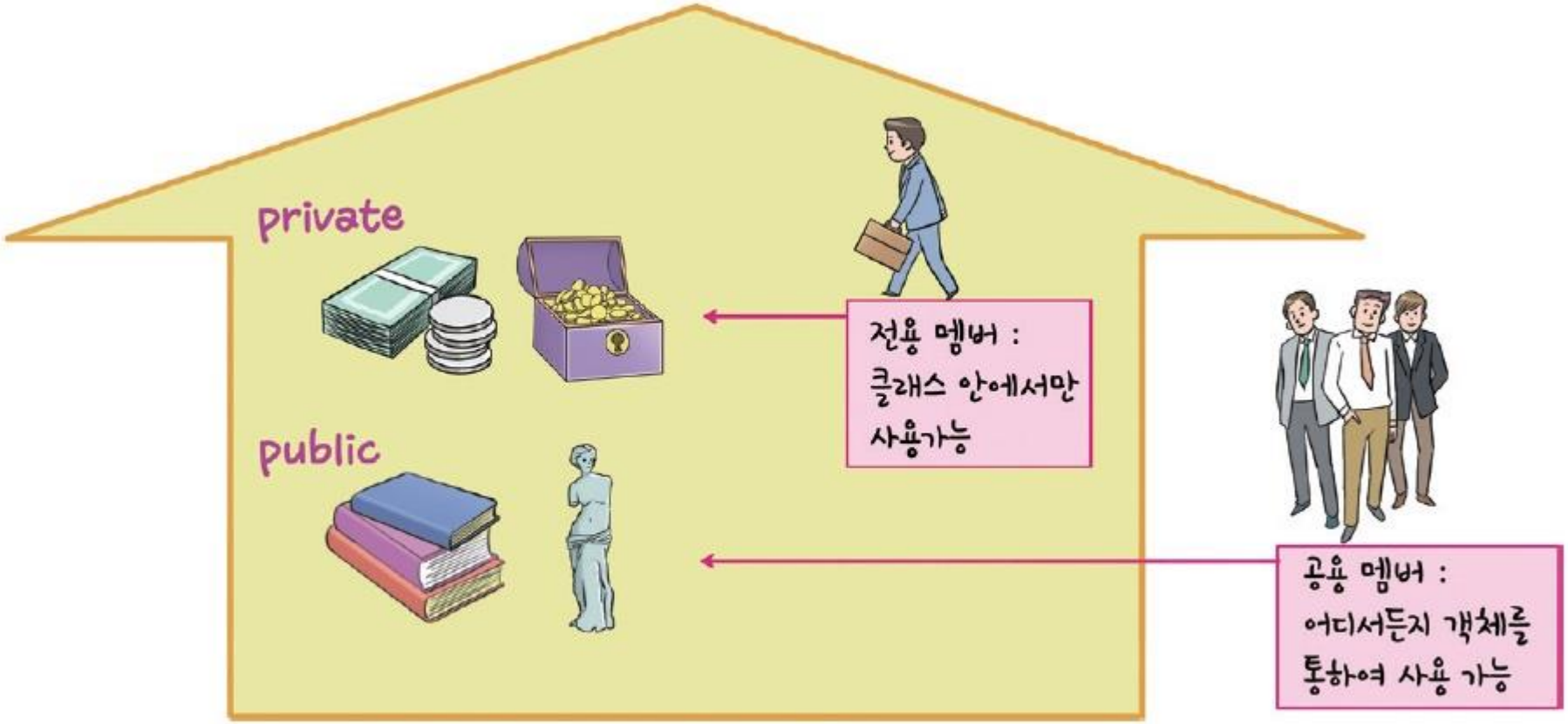


그림 5.2 접근 지정자

접근 지정자

- 접근 지정자를 통한 접근 제어
- 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

- 캡슐화 목적
  - 객체 보호

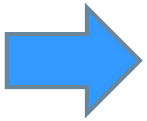
- C++에서 객체의 캡슐화 전략
  - 객체의 상태를 나타내는 데이터 멤버(멤버 변수)에 대한 보호
  - 중요한 멤버는 다른 클래스나 객체에서 접근할 수 없도록 보호
  - 외부와의 인터페이스를 위해서 일부 멤버는 외부(또는 특정적으로 제한하여)에 접근 허용

디폴트 접근

- 디폴트 접근은 private
  - vs Java?

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();
};
```

멤버 변수는 private으로 지정


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

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

멤버 변수 보호받지 못함

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

(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;
}
```

멤버 변수 보호받고 있음

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

(b) 멤버 변수를 private으로 선언한 바람직한 사례

접근자와 설정자

- 외부에서 멤버 변수에 대한 접근을 제공
- 이러한 접근은 접근자(getter)와 설정자(setter)로 구성
  - Getter는 접근을 제공하고,
  - Setter는 설정을 제공
  - 이들은 보통 클래스 내부에 정의 (그 크기가 작으므로)

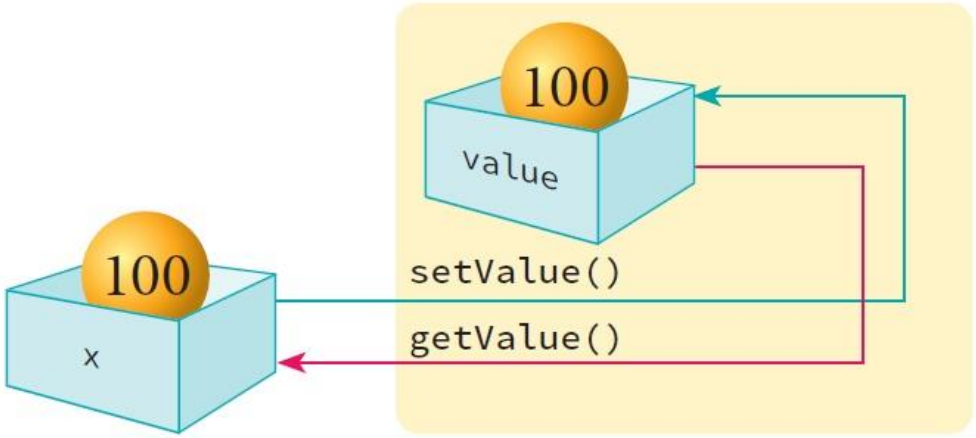


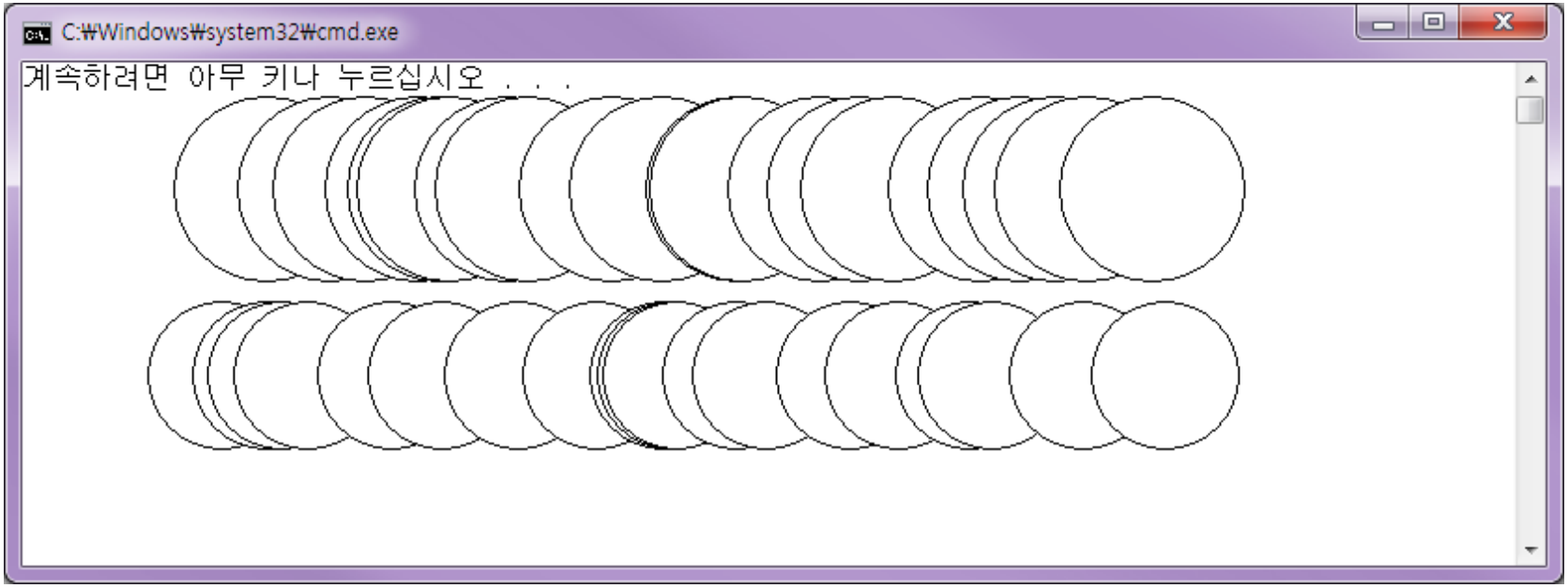
그림 5.3 접근자와 설정자는 멤버 변수의 접근을 제한한다.

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

class Time {
public:
        Time(int h, int m);
        void inc_hour();
        void print();

        int getHour() { return hour; }
        int getMinute() { return minute; }
        void setHour(int h) { hour = h; }
        void setMinute(int m) { minute = m; }

private:
        int hour;                       // 0-23
        int minute;                     // 0-59
};
```

```
int main()
{
        Time a{ 0, 0 };

        a.setHour(6);
        a.setMinute(30);

        a.print();
        return 0;
}
```

Lab: 원들의 경주

- 지금까지 학습한 내용을 바탕으로 “원들의 경주 “ 게임을 다시 작성하여 보자. 두 개의 원을 생성한 후에 난수를 발생하여 원들을 움직인다.



Lab: Solution

```
#include <iostream>
#include <windows.h>
using namespace std;

class Circle {
public:
        Circle(int xval, int yval, int r);
        void draw();
        void move();
private:
        int x, y, radius;
};

Circle::Circle(int xval, int yval, int r) : x{ xval }, y{ yval }, radius{ r }
{
}
```

```
void Circle::draw() {
        HDC hdc = GetWindowDC(GetForegroundWindow());
        Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);
}

void Circle::move() {
        x += rand() % 50;
}

int main()
{
        Circle c1{ 100, 100, 50 };
        Circle c2{ 100, 200, 40 };

        for (int i = 0; i < 20; i++) {
                c1.move();
                c1.draw();
                c2.move();
                c2.draw();
                Sleep(1000);
        }
        return 0;
}
```

```
class Circle {
public:
        Circle(int xval, int yval, int r);
        void draw();
        void move();
private:
        int x, y, radius;
};

Circle::Circle(int xval, int yval, int r)
        : x{ xval }, y{ yval }, radius{ r } {
}
```

객체와 함수

- 객체가 함수의 매개 변수로 전달될 때
- 객체의 참조나 주소가 함수의 매개 변수로 전달될 때
- 함수가 객체를 반환할 때

객체가 매개 변수로 전달될 때

- 객체를 복사하여 전달한다.
- 함수에서 객체를 변경하여도 실 매개변수 객체의 상태는 반영되지 않는다.
- 객체를 누가 복사하는가? (9장 복사생성자 참고)

예제

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

class Pizza {
public:
        Pizza(int s) : size(s) { }
        int size;
};

void makeDouble(Pizza p){
        p.size *= 2;
}

int main(){
        Pizza pizza(10);

        makeDouble(pizza);
        cout << pizza.size << "인치 피자" << endl;

        return 0;
}
```

객체의 참조가 매개 변수로 전달될 때

- 대응 형식 매개변수 타입이 객체에 대한 참조 타입인 경우
- 함수에서 객체를 변경하면 실 매개변수 객체의 상태에 반영된다.
- 객체의 참조를 전달하는 것이 복사하는 것보다 더 효율적

예제: 객체의 참조가 매개 변수로 전달 경우

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

class Pizza {
public:
        Pizza(int s) : size(s) { }
        int size;
};

void makeDouble(Pizza& p) { // p의 타입은?
        p.size *= 2;
}

int main() {
        Pizza pizza(10);
        makeDouble(pizza);
        cout << pizza.size << "인치 피자" << endl;

        return 0;
}
```

함수가 객체를 반환할 때

- 객체가 복사되어 반환된다.
- 누가 객체를 복사하는가?

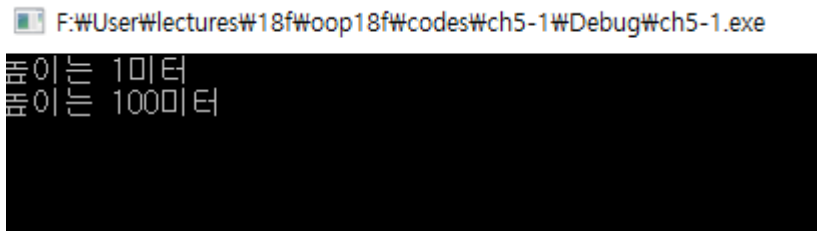
예제

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

class Pizza {
public:
        Pizza(int s) : size(s) { }
        int size;
};

Pizza createPizza() {
        Pizza p(10);
        return p;
}

int main() {
        Pizza pizza = createPizza();
        cout << pizza.size << "인치 피자" << endl;

        return 0;
}
```

실습

- main의 실행 결과가 다음과 같도록 Tower 클래스를 작성하라.

```
int main() {
   Tower myTower;
   Tower seoulTower(100);

   cout << "높이는 " << myTower.getHeight() << "미터" << endl;
   cout << "높이는 " << seoulTower.getHeight() << "미터" << endl;
}
```



실습

- 날짜를 다루는 Date 클래스를 작성하라. Date를 이용하는 main()과 실행 결과는 다음과 같다.

```
int main() {
   Date birth(2014, 3, 20);
   Date independenceDay("1945/8/15");

   independenceDay.show();
   cout << birth.getYear() << ',' << birth.getMonth() << ',' <<     birth.getDay()
<< endl;
}
```