06

함수 중복과 static 멤버

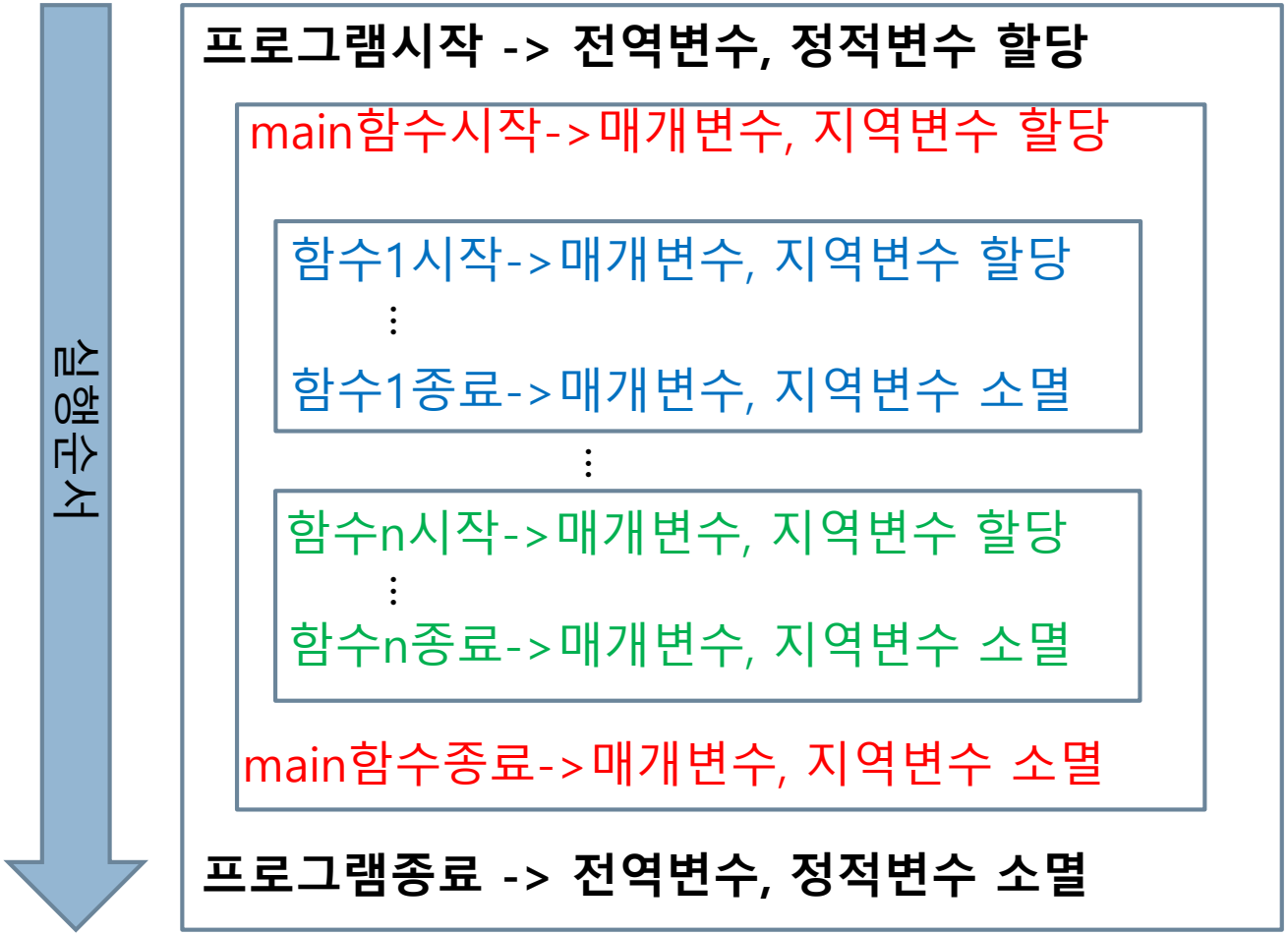
학습 목표

1. 함수 중복의 개념을 이해하고, 중복 함수를 작성할 수 있다.
2. 디폴트 매개 변수를 이해하고 작성할 수 있다.
3. 함수 중복 시 발생하는 모호성의 경우를 판별할 수 있다.
4. static 속성으로 선언된 멤버의 특성을 이해하고, static 속성을 활용할 수 있다.

static 변수

	지역변수	전역변수	정적지역변수
선언위치	함수 내부	함수외부	함수내부
접근범위	함수내부	모든 함수	함수내부
생존기간	함수호출~종료	프로그램시작~종료	프로그램시작~종료
자동초기화	X	O	O
초기값	쓰레기값	0	0
장단점	함수내부에서만 접근 가능-> 변수 오염방지	함수들 사이에 데이터 교환편리, 버그의 원인, 디버깅 어려움	접근범위를 함수내부로 제한하면서 함수 종료후에도 값이 존재

메모리 할당 및 소멸순서


실행순서
프로그램시작 -> 전역변수, 정적변수 할당
main함수시작->매개변수, 지역변수 할당
함수1시작->매개변수, 지역변수 할당
함수1종료->매개변수, 지역변수 소멸
함수n시작->매개변수, 지역변수 할당
함수n종료->매개변수, 지역변수 소멸
main함수종료->매개변수, 지역변수 소멸
프로그램종료 -> 전역변수, 정적변수 소멸

static 멤버와 non-static 멤버

- static 멤버
  - 프로그램이 시작할 때 생성(main함수가 실행되기 이전)
  - 객체 생성전에 메모리에 생성됨
  - 프로그램 종료시 소멸
  - 클래스 당 하나만 별도의 공간에 생성, 클래스 멤버라고 불림
  - 클래스의 모든 객체들이 공유하는 멤버
- non-static 멤버
  - 객체가 생성될 때 함께 생성, 객체 소멸시 사라짐
  - 객체마다 별도로 생성
  - 인스턴스(객체) 멤버라고 불림

static 멤버 선언

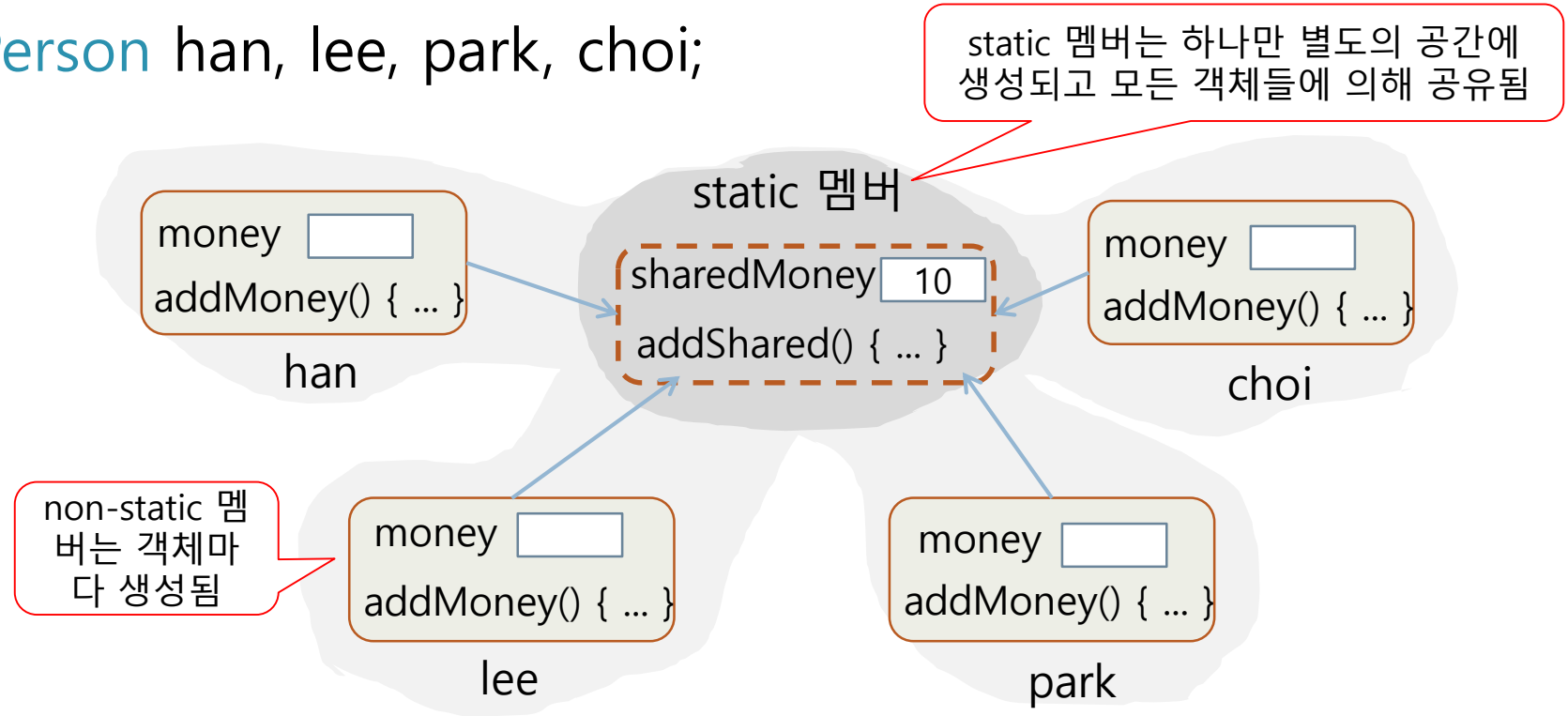
- 클래스 선언부에서 멤버 함수나 멤버 변수 선언 앞에 static 지정자 추가
- static 멤버들은 private, public, protected 등 모든 접근지정도 가능
- **static 멤버 변수는 클래스 외부에 전역변수로 선언해야 함**
- **static 멤버함수의 구현부와 static 멤버변수의 외부선언에는 static을 붙이면 안됨**

static 멤버 선언


```
class Person {
public:
	double money;                       //non-static 멤버 변수 선언
	void addMoney(int money);           //non-static 멤버 함수 선언
	static int sharedMoney;             //static 멤버 변수 선언
	static void addShared(int n);       //static 멤버 함수 선언
};
int Person::sharedMoney = 10;           //전역 공간에 선언해야 함
void Person::addMoney(int money) {      //non-static 멤버 함수 정의
	…
}
void Person::addShared(int n) {         //static 멤버 함수 정의
	…
}
```

static 멤버와 non-static 멤버의 관계

Person han, lee, park, choi;



- han, lee, park, choi 등 4 개의 Person 객체 생성
- **sharedMoney와 addShared() 함수는 하나만 별도의 공간에 생성되고 4개의 객체들의 의해 공유됨**
- sharedMoney와 addShared() 함수는 han, lee, park. choi 객체들의 멤버임

static 멤버와 non-static 멤버 비교

항목	non-static 멤버	static 멤버

선언 사례	class Sample {
	  int n;
	  void f();
	};
		class Sample {
		  static int n;
		  static void f();
		};

공간 특성	멤버는 객체마다 별도 생성 • 인스턴스 멤버라고 부름	멤버는 클래스 당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간에 생성 • 클래스 멤버라고 부름
시간적 특성	객체와 생명을 같이 함 • 객체 생성 시에 멤버 생성 • 객체 소멸 시 함께 소멸 • 객체 생성 후 객체 사용 가능	프로그램과 생명을 같이 함 • 프로그램 시작 시 멤버 생성 • 객체가 생기기 전에 이미 존재 • 객체가 사라져도 여전히 존재 • 프로그램이 종료될 때 함께 소멸
공유의 특성	공유되지 않음 • 멤버는 객체 별로 따로 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

static 멤버 사용 : 객체의 멤버로 접근

- static 멤버는 객체명이나 객체 포인터로 접근 가능

```
객체명.static멤버
객체포인터->static멤버
```

```
Person lee;
lee.sharedMoney = 500;          // 객체명 이용
Person* p = &lee;
p->addShared(200);              // 객체포인터 이용
```

static 멤버 사용 : 객체의 멤버로 접근


```
#include <iostream>
using namespace std;
class Person {
public:
	double money;                    // 개인 소유의 돈
	void addMoney(int money) {
		this->money += money;
	}
	static int sharedMoney;          // 공금
	static void addShared(int n) {
		sharedMoney += n;
	}
};
int Person::sharedMoney = 10;        // static 변수 생성. 전역 공간에 생성
```

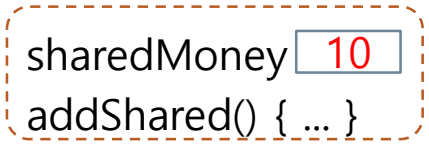
static 멤버 사용 : 객체의 멤버로 접근


```
int main() {
	Person han;
	han.money = 100;		// han의 개인 돈=100
	han.sharedMoney = 200;		// static 멤버 접근, 공금=200

	Person lee;
	lee.money = 150;		// lee의 개인 돈=150
	lee.addMoney(200);		// lee의 개인 돈=350
	lee.addShared(200);		// static 멤버 접근, 공금=400

	cout << han.money << ' ' << lee.money << endl;
	cout << han.sharedMoney << ' ' << lee.sharedMoney << endl;
}
```

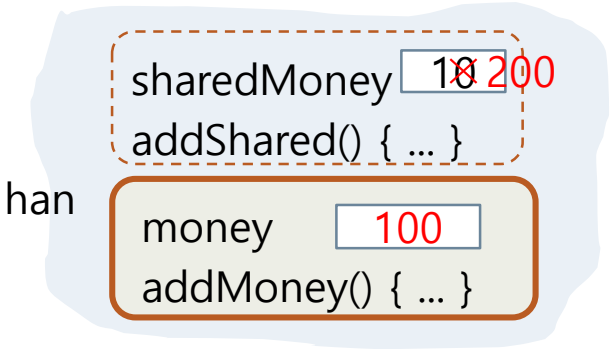
```
100 350
400 400
```

static 멤버 사용 : 객체의 멤버로 접근

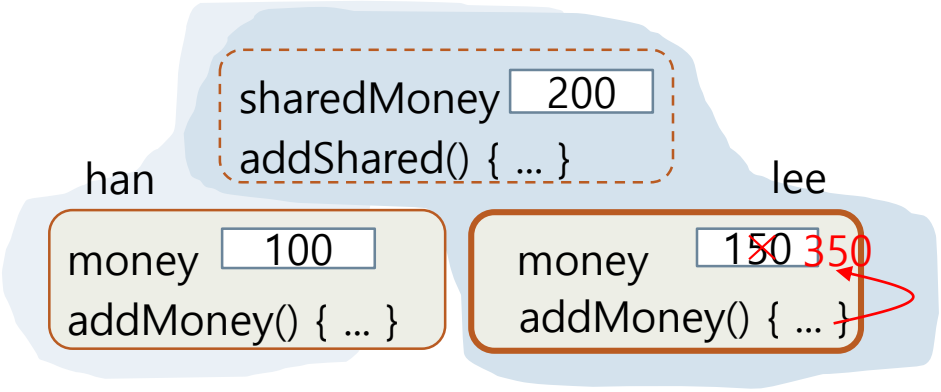
main()이 시작하기 직전



Person han;
han.money = 100;
han.sharedMoney = 200;

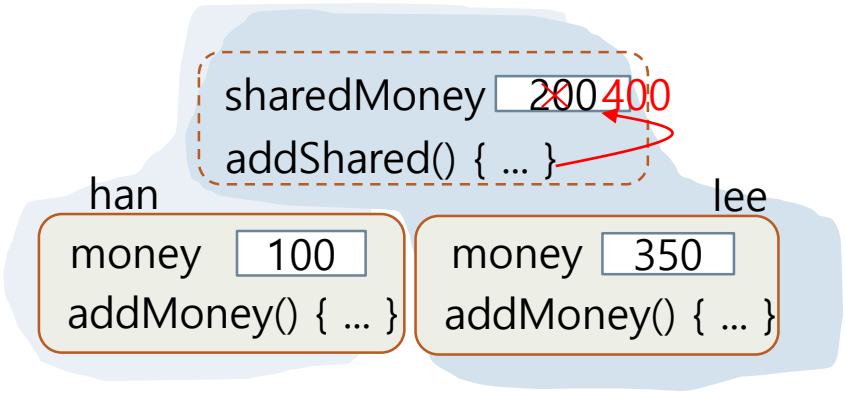


Person lee;
lee.money = 150;
lee.addMoney(200);



static 멤버 사용 : 객체의 멤버로 접근

lee.addshared(200);



static 멤버사용: 클래스명과 범위지정연산자(::)로 접근

- 클래스 이름과 범위 지정 연산자(::)로 접근 가능
  - static 멤버는 클래스마다 오직 한 개만 생성되기 때문

```
클래스명::static멤버
```

```
Person::sharedMoney = 200;      // 클래스명으로 접근
Person::addShared(200);         // 클래스명으로 접근

han.sharedMoney = 200;  <->  Person::sharedMoney = 200;
lee.addShared(200);     <->  Person::addShared(200);
```

- non-static 멤버는 클래스명으로 접근 불가

```
Person::money = 100;            // 컴파일 오류
Person::addMoney(200);          // 컴파일 오류
```

static 멤버사용 예


```
#include <iostream>
using namespace std;
class Person {
public:
	double money;                    // 개인 소유의 돈
	void addMoney(int money) {
		this->money += money;
	}
	static int sharedMoney;          // 공금
	static void addShared(int n) {
		sharedMoney += n;
	}
};
int Person::sharedMoney = 10;    // static변수생성. 10으로 초기화
```

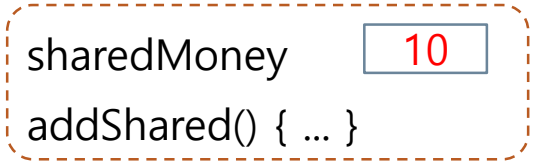
static 멤버사용 예


```
int main() {
    Person::addShared(50);       // static멤버접근, 공금=60
    cout << Person::sharedMoney << endl;

    Person han;
    han.money = 100;
    han.sharedMoney = 200;              // static 멤버 접근, 공금=200

    Person::sharedMoney = 300;          // static 멤버 접근, 공금=300
    Person::addShared(100);             // static 멤버 접근, 공금=400
    cout << han.money << ' ' << Person::sharedMoney << endl;
}
```

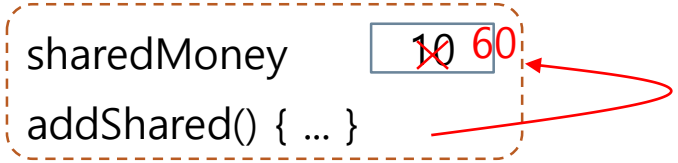
```
60
100 400
```

static 멤버사용 예

main()이 시작하기 직전

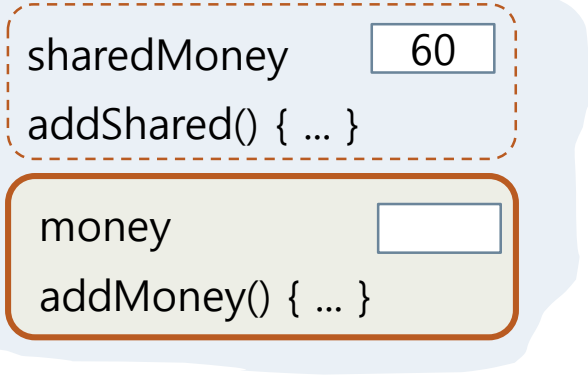


Person::addShared(50);



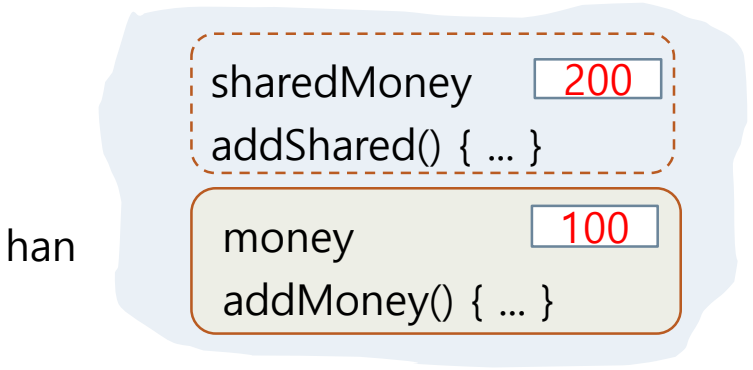
Person han;

han

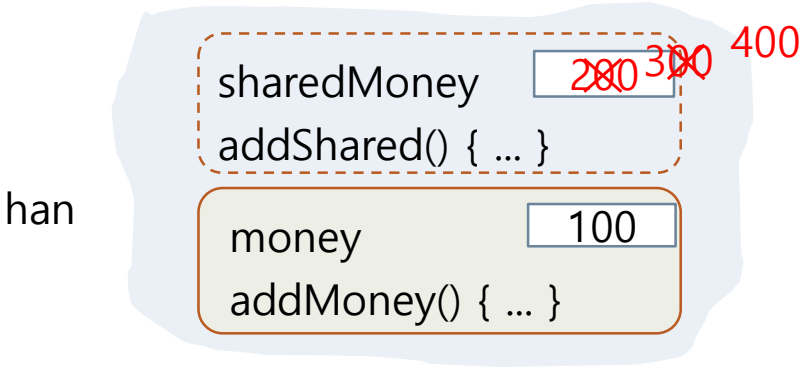


static 멤버사용 예

han.money = 100;
han.sharedMoney = 200;



Person::sharedMoney = 300;
Person::addShared(100);



static 멤버 활용

- static 멤버의 사용 목적
  - 전역 변수나 전역 함수를 클래스 안에 캡슐화(예제 6-10)
    - 전역 변수나 전역 함수를 가능한 사용하지 않는 것이 좋음
    - 전역 변수나 전역 함수를 static으로 선언하여 클래스 멤버로 선언
  - 객체 사이에 공유 변수를 만들고자 할 때(예제 6-11)
    - static 멤버를 선언하여 모든 객체들이 공유

예제 6-10 static 멤버를 가진 Math 클래스 작성

- 아래 코드를 static 멤버를 가진 Math 클래스로 작성하고 멤버 함수를 호출하라.

```
#include <iostream>
using namespace std;
int abs(int a) { return a > 0 ? a : -a; }
int max(int a, int b) { return  a > b) ? a : b; }
int min(int a, int b) { return (a > b) ? b : a; }
int main() {
      cout << abs(-5) << endl;
      cout << max(10, 8) << endl;
      cout << min(-3, -8) << endl;
}
```

(a) 전역 함수들을 가진 좋지 않은 코딩 사례

예제 6-10 static 멤버를 가진 Math 클래스 작성


```
#include <iostream>
using namespace std;
class Math {
public:
	static int abs(int a) { return a > 0 ? a : -a; }
	static int max(int a, int b) { return (a > b) ? a : b; }
	static int min(int a, int b) { return (a > b) ? b : a; }
};
int main() {
	cout << Math::abs(-5) << endl;
	cout << Math::max(10, 8) << endl;
	cout << Math::min(-3, -8) << endl;
}
```

```
5
10
-8
```

(b) Math 클래스를 만들고 전역 함수들을 static 멤버로 캡슐화한 프로그램

예제6–11 static 멤버를 공유목적으로 사용하는 예

- 생성된 Circle 객체의 개수를 기억하는 변수를 정적변수로 선언

```
#include <iostream>
using namespace std;
class Circle {
private:
	static int numOfCircles;
	int radius;
public:
	Circle(int r = 1);
	~Circle() { numOfCircles--; }	// 생성된 원의 개수 감소
	double getArea() { return 3.14 * radius * radius; }
	static int getNumOfCircles() { return numOfCircles; }
};
Circle::Circle(int r) {
	radius = r;
	numOfCircles++;			// 생성된 원의 개수 증가
}
int Circle::numOfCircles = 0;		// 0으로 초기화
```

```
생존하고 있는 원의 개수 = 10
생존하고 있는 원의 개수 = 0
생존하고 있는 원의 개수 = 1
생존하고 있는 원의 개수 = 2
```

예제6–11 static 멤버를 공유목적으로 사용하는 예


```
int main() {
	Circle* p = new Circle[10];	// 10개의 생성자 실행
	cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles()
		<< endl;

	delete [] p;				// 10개의 소멸자 실행
	cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles()
		<< endl;

	Circle a;					// 생성자 실행
	cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles()
		<< endl;

	Circle b;					// 생성자 실행
	cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles()
		<< endl;
}
```

static 멤버 함수는 static 멤버만 접근 가능

- static 멤버 함수가 접근할 수 있는 것
  - static 멤버 함수
  - static 멤버 변수
  - 함수 내의 지역 변수
- static 멤버 함수는 non-static 멤버에 접근 불가
  - 객체가 생성되지 않은 시점에서 static 멤버 함수가 호출될 수 있기 때문
  - this 포인터 사용불가

static 멤버 함수가 non-static 멤버변수를 접근


```
class PersonError {
      int money;
public:
      static int getMoney() { return money; }
      void setMoney(int money) {                    // 정상 코드
            this->money = money;
      }
};
int main() {
      int n = PersonError::getMoney();              // 오류
      PersonError errorKim;
      errorKim.setMoney(100);
}
```

컴파일 오류 -> static 멤버 함수는 non-static 멤버에 접근할 수 없음.

static 멤버 함수가 non-static 멤버 변수를 접근

main()이 시작하기 전

```
static int getMoney() {
   return money;
}
```

money는 아직 생성되지 않았음.

n = PersonError::getMoney();

```
static int getMoney() {
   return money;
}
```

생성되지 않는 변수를 접근하게 되는 오류를 범함

PersonError errorKim;

errorKim

```
static int getMoney() {
   return money;
}
```

money

setMoney() { ... }

errorKim 객체가 생길 때 money가 비로소 생성됨

non-static 멤버 함수는 static에 접근 가능


```
class Person {
public:
	double money;			// non-static, 개인 소유의 돈
	static int sharedMoney;		// static, 공금
	....
	int total() { return money + sharedMoney; }
};
```

non-static 함수는 non-static이나 static 멤버에 모두 접근 가능

static 멤버 함수는 this 사용 불가

- static 멤버 함수는 객체가 생기기 전부터 호출 가능
  - static 멤버 함수에서 this 사용 불가

```
class Person {
public:
      double money;                  // 개인 소유의 돈
      static int sharedMoney;        // 공금
      ....
      static void addShared(int n) {       // static 함수에서 this 사용 불가
            this->sharedMoney + = n;   // this를 사용하므로 컴파일 오류
      }
};
```

sharedMoney += n;
으로 하면 정상 컴파일

실습과제1

- static 멤버변수는 클래스 선언부에 선언하고 외부에서 다시 선언하는 이유를 설명하라.
- static 멤버함수의 클래스 구현부에 static 키워드를 붙이면 안되는 이유를 설명하라.

실습과제2

- 배열 데이터 중에서 최대값과 최소값을 구하는 함수를 포함하는 MyMath 클래스를 선언하시오.(예제 6-10참고)

```
#include <iostream>
using namespace std;
// 클래스 선언 추가
int main() {
	int x[5] = { 20, 30, -5, 2, -30 };
	cout << "최대값은 :" << MyMath::GetMax(x, 5) << endl;
	cout << "최소값은 :" << MyMath::GetMin(x, 5) << endl;
	return 0;
}
```

```
최대값은 : 30
최소값은 : -30
```

실습과제3

- 아래 결과처럼 나오는데 필요한 Triangle클래스의 멤버 함수를 작성하시오.(예제 6-11참고)

```
생성된 삼각형의 개수: 5
생성된 삼각형의 개수: 15
```

```
#include <iostream>
using namespace std;
// 클래스 선언 추가
int main() {
	Triangle* tri1 = new Triangle[5];
	cout << "생성된 삼각형의 개수 :" << Triangle::getNumofTriangle()
	<< endl;
	delete [ ] tri1;
	Triangle tri2[15];
	cout << "생성된 삼각형의 개수 :" << Triangle:: getNumofTriangle()
	<< endl;
	return 0;
}
```

실습문제4

- 313~318페이지 문제 중에서 1,2,3,4,5번을 풀어서 제출하시오.

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
```