

함수 중복

1

□ 함수 중복

▣ 동일한 이름의 함수가 공존

- 다형성
- C 언어에서는 불가능

▣ function overloading

▣ 함수 중복이 가능한 범위

- 보통 함수들 사이
- 클래스의 멤버 함수들 사이
- 상속 관계에 있는 기본 클래스와 파생 클래스의 멤버 함수들 사이

□ 함수 중복 성공 조건

- ▣ 중복된 함수들의 이름 동일
- ▣ 중복된 함수들의 매개 변수 타입이 다르거나 개수가 달라야 함
- ▣ 리턴 타입은 함수 중복과 무관

함수 중복 성공 사례

2

```
int sum(int a, int b, int c) {  
    return a + b + c;  
}  
  
double sum(double a, double b) {  
    return a + b;  
}  
  
int sum(int a, int b) {  
    return a + b;  
}
```

성공적으로 중복된 sum() 함수들

```
int main(){  
    cout << sum(2, 5, 33);  
  
    cout << sum(12.5, 33.6);  
  
    cout << sum(2, 6);  
}
```

중복된 sum() 함수 호출.
컴파일러가 구분

함수 중복 실패 사례

3

- 리턴 타입이 다르다고 함수 중복이 성공하지 않는다.

```
int sum(int a, int b) {  
    return a + b;  
}  
double sum(int a, int b) {  
    return (double)(a + b);  
}
```

함수 중복 실패

```
int main() {  
    cout << sum(2, 5);  
}
```

?

컴파일러는 어떤 sum()
함수를 호출하는지 구
분할 수 없음

함수 중복의 편리함

4

- 동일한 이름을 사용하면 함수 이름을 구분하여 기억할 필요 없고, 함수 호출을 잘못하는 실수를 줄일 수 있음

```
void msg1() {  
    cout << "Hello";  
}  
void msg2(string name) {  
    cout << "Hello, " << name;  
}  
void msg3(int id, string name) {  
    cout << "Hello, " << id << " " << name;  
}
```

(a) 함수 중복하지 않는 경우



```
void msg() {  
    cout << "Hello";  
}  
void msg(string name) {  
    cout << "Hello, " << name;  
}  
void msg(int id, string name) {  
    cout << "Hello, " << id << " " << name;  
}
```

(b) 함수 중복한 경우

함수 중복하면 함수 호출의 편리함.
오류 가능성 줄임

생성자 함수 중복

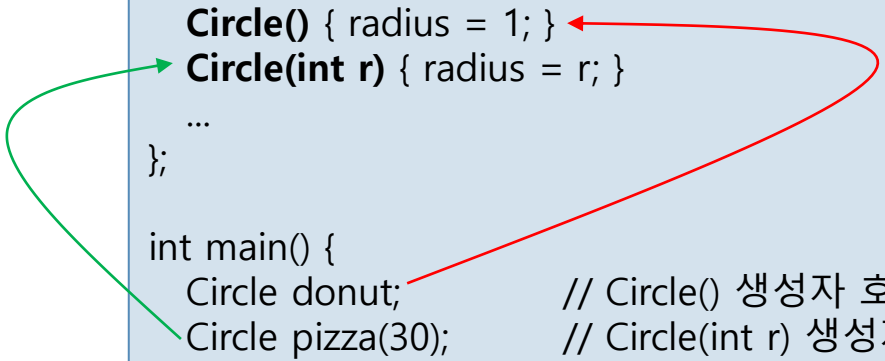
5

□ 생성자 함수 중복 가능

▣ 생성자 함수 중복 목적

- 객체 생성시, 매개 변수를 통해 다양한 형태의 초기값 전달

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    ...  
};  
  
int main() {  
    Circle donut;           // Circle() 생성자 호출  
    Circle pizza(30);       // Circle(int r) 생성자 호출  
}
```



string 클래스의 생성자 중복 사례

6

```
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("서울시 강남구 역삼동");  
string copyAddress(address); // address의 문자열을 복사한 별도의 copyAddress 생성
```

소멸자 함수 중복

7

- 소멸자 함수 중복 불가
 - ▣ 소멸자는 매개 변수를 가지지 않음
 - ▣ 한 클래스 내에서 소멸자는 오직 하나만 존재

디폴트 매개 변수

8

- 디폴트 매개 변수(default parameter)
 - ▣ 매개 변수에 값이 넘어오지 않는 경우, 디폴트 값을 받도록 선언된 매개 변수
 - ‘매개 변수 = 디폴트값’ 형태로 선언

- 디폴트 매개 변수 선언 사례

```
void star(int a=5); // a의 디폴트 값은 5
```

- 디폴트 매개 변수를 가진 함수 호출

```
star(); // 매개 변수 a에 디폴트 값 5가 전달됨. star(5);와 동일  
star(10); // 매개 변수 a에 10을 넘겨줌
```


디폴트 매개 변수에 관한 제약 조건

9

- ▣ 디폴트 매개 변수는 보통 매개 변수 앞에 선언될 수 없음
 - 디폴트 매개 변수는 끝 쪽에 몰려 선언되어야 함

컴파일 오류

```
void calc(int a, int b=5, int c, int d=0); // 컴파일 오류  
void sum(int a=0, int b, int c); // 컴파일 오류
```

```
void calc(int a, int b=5, int c=0, int d=0); // 컴파일 성공
```

함수 중복의 모호성

10

- 함수 중복이 모호하여 컴파일러가 어떤 함수를 호출하는지 판단하지 못하는 경우
 - ▣ 형 변환으로 인한 모호성
 - ▣ 참조 매개 변수로 인한 모호성
 - ▣ 디폴트 매개 변수로 인한 모호성

생성자 함수의 중복 간소화

11

```
class MyVector{  
    int *p;  
    int size;  
public:  
    MyVector() {  
        p = new int [100];  
        size = 100;  
    }  
    MyVector(int n) {  
        p = new int [n];  
        size = n;  
    }  
    ~MyVector() { delete [] p; }  
};
```

정답

```
#include <iostream>  
using namespace std;  
  
class MyVector{  
    int *p;  
    int size;  
public:  
    MyVector(int n=100) {  
        p = new int [n];  
        size = n;  
    }  
    ~MyVector() { delete [] p; }  
};  
  
int main() {  
    MyVector *v1, *v2;  
    v1 = new MyVector(); // 디폴트로 정수 100개의 배열 동적 할당  
    v2 = new MyVector(1024); // 정수 1024개의 배열 동적 할당  
  
    delete v1;  
    delete v2;  
}
```

static 멤버와 non-static 멤버의 특성

12

□ static

▣ 변수와 함수에 대한 기억 부류의 한 종류

- 생명 주기 - 프로그램이 시작될 때 생성, 프로그램 종료 시 소멸
- 사용 범위 - 선언된 범위, 접근 지정에 따름

□ 클래스의 멤버

▣ static 멤버

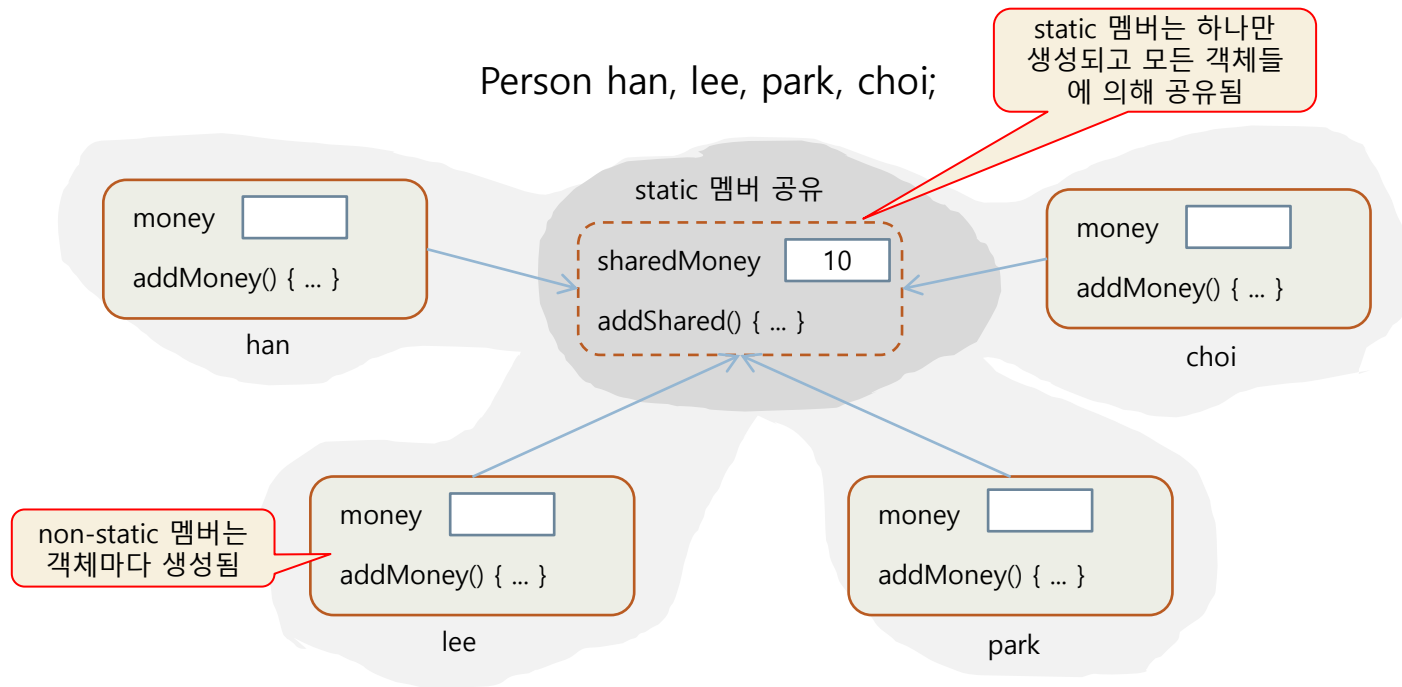
- 프로그램이 시작할 때 생성
- 클래스 당 하나만 생성, 클래스 멤버라고 불림
- 클래스의 모든 인스턴스(객체)들이 공유하는 멤버

▣ non-static 멤버

- 객체가 생성될 때 함께 생성
- 객체마다 객체 내에 생성
- 인스턴스 멤버라고 불림

static 멤버와 non-static 멤버의 관계

13



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

static 멤버와 non-static 멤버 비교

14

항목	non-static 멤버	static 멤버
선언 사례	<pre>class Sample { int n; void f(); };</pre>	<pre>class Sample { static int n; static void f(); };</pre>
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 : 클래스명과 범위 지정 연산자(::)로 접근

15

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

클래스명::static멤버

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

- ▣ non-static 멤버는 클래스 이름을 접근 불가

Person::money = 100; // 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가
Person::addMoney(200); // 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가

static 활용

16

□ static의 주요 활용

- ▣ 전역 변수나 전역 함수를 클래스에 캡슐화
 - 전역 변수나 전역 함수를 가능한 사용하지 않도록
 - 전역 변수나 전역 함수를 static으로 선언하여 클래스 멤버로 선언
- ▣ 객체 사이에 공유 변수를 만들고자 할 때
 - static 멤버를 선언하여 모든 객체들이 공유

static 멤버 함수는 static 멤버만 접근 가능

17

- ▣ 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

클래스와 `const`

- **멤버 변수의 상수화, 그리고 초기화**
 - ▣ **멤버 이니셜라이저(member initializer)**

```
Student(...) : id(_id)
```

- ▣ 생성자는 메모리할당 후 대입되므로 상수멤버 대입 불가 함. 따라서 멤버 이니셜라이저로 초기화한다.

- **`const` 멤버 함수**
 - ▣ 멤버 변수의 값 변경 허용 않는다.
 - ▣ 멤버 변수 값의 변경에 대한 기회제공도 불가

클래스와 const

□ const 객체

- ▣ 데이터의 변경이 허용되지 않는 객체
- ▣ const 함수 이외에는 호출 불가
- ▣ 상수화된 함수는 멤버의 주소 반환불가

□ const와 함수 오버로딩

- ▣ const도 함수 오버로딩 조건에 포함

```
void function(int n) const  
{ . . . . . }  
void function(int n) { . . . . . }
```

- ▣ Mutable: 상수화된 메서드에서 mutable로 선언된 멤버는 수정이 가능하다.