HW_03_19기_예비단원_구도연

<클래스와 상속>

1. 클래스

: 데이터와 함수를 하나의 묶음으로 취급하는 사용자 정의 데이터 타입.
구조체와 유사한 형식으로 쉽게 이해하기 위한 예로 다음을 이야기할 수 있다.
현실의 특정 사물에 빗대어 클래스를 이야기하면, 사물의 상태 및 특성과 관련된 기능을 넣는 공간 혹은 틀이라고 생각할 수 있다.

예를 들어 자동차 라는 커다란 이름 안에 자동차의 상태와 특성(e.g. 차량번호, 연료....)을 정리해 넣은 공간이라고 이야기할 수 있다.

+클래스가 C언어의 구조체와 유사한 이유.
: 구조체의 작성 방법과 클래스의 작성 방법이 거의 유사하기 때문.

클래스 선언 방법.

```
:
  class 클래스명{
     접근 지정자;
     변수 선언;   // 클래스 안에 선언하는 변수 및 함수들은 멤버라고 부른다.
     함수 선언; // 이때, 변수는 데이터 멤버. 함수는 멤버 함수로 구분해서 부름.
          ...
  };
```

위와 같은 방식으로 선언을 하며 구조체와의 차이점은 struct 키워드가 아닌 class 키워드를 사용하는 점이다.

++ 함수 선언 시 주의 사항. 멤버 함수의 본체는 클래스 바깥에서 정의.

사용 예시

```
Class Dog{
    Public:
    // 멤버 변수
    String name;
    Int age;

    Void bark(){
        cout << "Bark!" << endl;
```

```
    }
};
```

⇨ Dog라는 클래스 정의한 것으로 Dog 형식의 객체를 만들 수 있음. 이 객체는 name, age 두 가지의 변수를 가지며 dark라는 함수를 상용할 수 있다.

멤버 변수와 멤버 함수.
: 멤버 변수는 클래스가 가지는 데이터를 정의함. 멤버 함수는 그 데이터를 다루는 방법을 정의하는 곳.

멤버 변수, 함수는 '접근 지시자'에 의해 제어되며 public, private, protected와 같은 접근 지시자를 사용하여 멤버들의 접근 권한을 설정할 수 있다.

++ public = 어떤 곳에서든 접근 가능
Private = 클래스 내부에서만 접근 가능
Protected = 클래스 내부와 해당 클래스를 상속받은 자식 클래스에서만 접근 가능

2. 상속
: 부모, 즉 기반 클래스에서 자식, 파생 클래스로 속성과 메서드를 전당하는 과정을 말함. 이를 통해 중복 코드를 최소화하고, 소프트웨어의 유지 관리를 용이하게 함.

사용 방법은 아래와 같다.

```cpp
// C++ 코드
class Animal {
public:
    void eat() {
        cout << "Eating...₩n";
    }

    void move() {
        cout << "Moving...₩n";
    }
};
```

위 코드가 Animal이라는 클래스를 정의할 때, 이 클래스는 eat와 move라는 두 개의 메서드를 가지고 있고 이를 사용해 객체를 생성할 수 있다.

이후 아래 코드와 같이 Animal 클래스의 객체인 cat을 생성하고 이를 통해 eat과 move 라는 메서드를 호출하는 예시이다.

```
Animal cat;
Cat .eat ();
Cat .move();
```

또한 아래의 코드와 같이 Animal 클래스를 상속받는 Cat클래스를 정의하여 이 클래스가 기존 클래스의 모든 속성과 메서드를 물려받아 사용할 수 있도록 할 수 있다.

즉 Cat 클래스의 객체는 eat, move, meow 메서드 모두 호출이 가능하다.

```
Class cat : public Animal {
Public:
    void meow(){
        Cout<<"Meowing...₩n";
    }
};

Cat kitty;
Kitty . eat();
Kitty . move();
Kitty . meow();
```

3. 부모 클래스(기반 클래스)와 자식클래스(파생 클래스)
   1) 부모 클래스
      : 상위 레벨에 있는 클래스로 특성을 물려준다.
   2) 자식 클래스
      : 하위 레벨의 클래스로 상위 레벨 클래스 내에 존재하는 모든 특성을 전달 받아 사용이 가능하다.

4. Friend, staic, const
   1) Friend
      : 두 개의 클래스가 있을 때, 서로 만들어진 모든 멤버 변수, 멤버 함수를 공유하기 위해서 사용하는 키워드로 a 객체가 b객체를 친구로 선언 하면 b객체는 a객체의 모든 변수와 함수값을 가져다 사용할 수 있다.

   2) Staic
      : Staic 키워드가 가지는 의미는 다음과 같다. 전역 변수 필드 내에 선언 시 선언된 파일 내에서만 참조를 허용하고, 함수 내에 선언되면 메모리 데이터 영역에 딱 한 번 생성과 동시에 초기화 되어 저장되며, 함수 내에서 그래도 지역적인 성격을 가지고 있지만, 함수가 종료되어도 소멸되지 않는다.

      Staic 멤버 변수는 클래스 변수라도 하며, 이는 일반 클래스의 멤버 변수와 달리 객체가 생성될 때마다 생성되지 않는다.

해당 클래스의 객체의 생성 여부에 상관하지 않고 프로그램이 시작되면 메모리 공간에 독립적으로 딱 하나만 할당되어 공유되는 변수로 생성과 소멸의 시기가 전역 변수와 동일하다.

++ Staic멤버는 객체에 종속되어 있는 변수가 아니라 외부에 존재하는 변수라는 이야기이며 이때 클래스 내부에 선언하여 멤버처럼 사용할 수 있는 이뉴는 객체에게 멤버 변수처럼 접근할 수 있는 권한만 부여되었기 때문이다.

Staic 멤버 함수 내에서는 Staic멤버 변수, Staic 멤버 함수만이 호출이 가능하다.

3) Const

: 객체에 Const 키워드를 선언하게 되면 이 객체를 대상으로 Const 멤버 함수만을 호출할 수 있다. 이는 Const 키워드가 데이터 변경을 허용하지 않음을 의미하기 때문이다.

함수오버로딩이 성립하기 위해서는 함수의 이름과 매개 변수 정보 등이 필요한데 거기에 추가로 Const 여부 또한 오버로딩의 조건이 될 수 있다.

5. 단일 상속, 다중 상속, 가상 상속

1) 기본 상속

: 기본 상속, 즉 단일 상속은 한 클래스의 속성과 메서드를 다른 클래스가 물려받는 것을 의미한다. 이를 통해 클래스 간의 계층적 관계 형성이 가능하며 상속 관계에서 물려주는 클래스를 상위 클래스 혹은 부모 클래스라 하고, 물려받는 클래스를 하위 클래스 또는 자식 클래스라고 한다.

C++ 에서 기본 상속 적용 시 상속 키워드를 사용해야 하며 이는 이어 상속 받을 클래스 명을 적고 뒤에 상속 방식을 적는 것으로 기본적 형식은 다음과 같다.

```
class DerivedClass : access-specifier BaseClass
```

+ 여기서 DerivedClass는 하위 클래스, BaseClass는 상위 클래스를 나타냄.
+ access-specifier은 접근 제어자로 앞서 나온 세 가지 중 하나를 사용할 수 있음.
 ++ public: 클래스의 어디에서든, 클래스의 외부에서도 접근이 가능함.
 ++ Protected: 같은 클래스와 하위 클래스에서만 접근이 가능함.
 ++ Private: 같은 클래스에서만 접근이 가능함.

```cpp
// C++ 코드
class BaseClass { // Base (parent) class
  public:
    void baseFunction() {
        cout << "This function is in the base class." << endl;
    }
};

class DerivedClass: public BaseClass { // Derived (child) class
};
```

```cpp
DerivedClass derivedObj;
derivedObj.baseFunction(); // This function is in the base class.
```

다음은 예제 코드로 DerivedClass는 BaseClass의 속성과 메서드를 상속 받았기 때문에 baseFunction 메서드 호출이 가능하다.

2) 다중 상속

: 하나의 클래스가 여러 개의 부모 클래스로부터 상속을 받을 수 있도록 해주는 것으로, 하나의 자식 클래스가 여러 부모 클래스의 속성과 메서드를 물려받을 수 있다.

```cpp
class DerivedClass: access-specifier BaseClass1, access-specifier BaseClass2, ... {
    // 클래스의 본문
};
```

다중 상속의 기본 문법은 위와 같으며 access-specifier은 접근 제어자로 앞서 나온 세 가지 중 하나를 사용할 수 있다.

다중 상속 사용에 있어 유의해야 할 점은 이름의 충돌로 이는 두 개 이상의 부모 클래스가 같은 이름의 메서드나 속성을 가지고 있을 때 발생함. 이때 자식 클래스에서 해당 이름을 사용하려고 하면 컴파일러는 어느 부모 클래스의 멤버를 참조해야 하는지 혼동하게 되며 이를 다이아몬드 문제라고 함.

해결 방법으로는 명시적인 범위 지정을 사용하는 것으로 자식 클래스에서 부모 클래스의 멤버를 참조할 때, 부모 클래스의 이름을 명시하여 어떤 부모 클래스의 멤버를 참조하였는지 명확하게 해주는 것이다.

3) 가상 상속

: 다중 상속이 가져올 수 있는 문제인 다이아몬드 문제를 해결하기 위한 특별한 형태의 상속이다. Virtual 키워드를 사용하여 상속을 선언하면 기본 클래스는 단 한 번만 상속되며, 이로 인해 기본 클래스의 인스턴스는 하위 클래스에서 공유되며 다이아몬드 문제가 해결되게 된다.

6. 가상 함수

   : 부모 클래스에서 상속받을 클래스에서 재정의할 것으로 기대하고 정의한 함수를 말한다. Virtual 이라는 예약어를 함수 앞에 붙여서 생성이 가능하며 이렇게 생성된 가상 함수는 파생 클래스에서 재정의 하면 이전에 정의된 내용들도 모두 새롭게 정의된 내용으로 바뀌게 된다.

7. 순수 가상 함수, 추상 클래스

   : virtual void food() = 0; 과 같은 형식을 가진 것을 순수 가상 함수라고 부르며, 함수의 정의가 이루어지지 않고 함수만 선언 한 것이다.
   또한 이렇게 선언된 순수 가상 함수가 있다면 이를 추상 클래스라고 부르며, 추상 클래스는 객체로 만들지 못하며 상속으로만 사용이 가능하다. 추상 클래스를 상속받은 자식 클래스는 무조건 해당 순수 가상 함수를 override 시켜줘야 한다.

   추상 클래스의 추상은 말 그대로 추상, 즉 상상과 같은 느낌의 단어로 동물이라는 클래스를 정의할 때, 여러 동물의 공통된 특징, 서로 다른 점 등 여러 요소를 나열 해두고 코드로 작성한다고 했을 때, 상속을 받지 않고 각 클래스 별로 정의한다면 내용을 일일이 다 적어줘야 하는 상황이 발생하며 이때 데이터를 빠뜨리거나 하는 실수가 생길 수 있는데 여기에 추상클래스를 사용하면 함수를 재정의하지 않았을 때 이를 오류로 판단하기 때문에 앞서 이야기한 실수를 방지할 수 있다.

8. +a

   1) '::'는 범위 결정 연산자로 :: 연산자를 이용해서 멤버가 어떤 클래스의 멤버 함수인지 알린다.