**개요**

**1. 객체지향프로그래밍이란**

**2. 객체란? 클래스란?**

**3. 객체지향프로그래밍의 특징(다형성, 상속성, 캡슐화(은닉화),추상화)**

**객체지향프로그래밍(Object - Oriented Programming, OOP)**

* **개념** : 처리하고자 하는 자료에 중점을 두고 프로그램을 **객체**라는 것으로 모델화하는 프로그래밍.
* **장점**

① 소프트웨어의 확장성 향상

② 재사용성 향상

③ 프로그래머의 생산성 향상

④ 유지보수 비용 절감

**Class 클래스**?

- **객체를 만들어 내기 위해 정의된 설계도, 틀**

- 클래스는 개체가 아님

- 멤버 변수와 멤버 함수 선언

- **같은 종류의 집단에 속하는 속성과 행위를 정의한 것**

- 클래스의 생성자 함수 : 객체가 선언될 때마다 자동적으로 호출

클래스와 같은 이름을 사용

반환 값은 없고, 파라미터는 가질 수 있다

클래스 내의 자료를 초기화

생성자 함수는 중복이 가능(오버로딩 가능)

정의하지 않으면 컴파일러에 의해 디폴트 생성자 만들어짐

- 클래스의 소멸자 함수 : 객체가 소멸될 때 자동으로 호출되어 할당받은 메모리 반환

클래스 이름 앞에 ~(틸드)를 붙이면 된다

반환 값과 파라미터를 가질 수 없다

하나의 클래스에는 하나의 소멸자만 가능(오버로딩 불가)

정의하지 않으면 디폴트 소멸자

**Object 객체**?

- 생성될 때 클래스의 모양을 가지고 탄생

- 멤버 변수와 멤버 함수로 구성

- 메모리에 생성되며, **클래스의 실체(instace)**라고도 부름

- 하나의 클래스 틀에서 여러 개의 객체 생성 가능

- **상위 틀래스의 속성을 가지고 있으면서 개별적인 특성과 행위 또한 가짐**

- 객체의 범위 :

① 전역 객체(global object) : main()위에서 선언,

생성자는 프로그램이 처음 시작할 때 한 번 실행되고 소멸자는 프로그램이 종료될 때 실행

② 지역 객체(local object) : 함수 내에서 선언,

생성자는 객체가 생성될 떄마다 실행되고 소멸자는 객체가 소멸될 때 실행

코드로 클래스와 객체를 보자.

class student{

private: //클래스 내에서만 엑세스 가능, 외부에서 접근 불가 friend클래스나 friend 함수에 의해서만 제어가능

int score;

protected: //외부에서 접근 불가능, 파생된 클래스에 의해서는 제어가능

// 클래스의 상속을 구현하는 과정에서 사용

public: //외부로부터 접근가능

//생성자 함수 중복가능 즉 오버로딩 가능

student(); //생성자

student(int x);//매개변수를 가지는 생성자

//혹은 student(int x){score = x;}라고 사용가능

~student(); //소멸자

void set(int a); //접근자

void show(); //멤버함수

int stu\_num;

};

student::student(){ //객체 초기화

score = 1;

}

student::student(int x){

score = x;

}

student::~student(){

cout << “End student” << endl;

}

void student::set(int a){

score = a;

}

void student::show(){

cout << score;

}

int main(){

student s1; //student class의 객체 s1

student s2(20); //혹은 student s2 = student(20);

s1.stu\_num = 12161616;

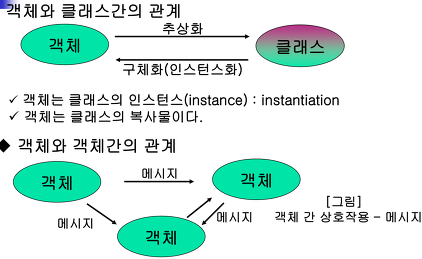
return 0;

}

Class 클래스 = variable 변수 + function 함수

**↑추상화            ↓구체화(인스턴트화)**

Object 객체 = Data 구성요소 + operation 행위



다시.. OOP로 돌아와서

* **OOP의 특성 (캡술화 + 은닉화, 추상화, 상속성, 다형성)**

**① 캡슐화(encapsulation)**

객체의 데이터를 외부에서 직접 접근하지 못하게 막고, 함수를 통해서만 접근이 가능하게 하는 작업.

데이터와 데이터를 처리하는 함수를 하나로 묶음

외부의 잘못된 사용으로 객체가 손상되지 않기 위해, 접근제한자(public, protected, private)사용해서 설정

장점

변경이 발생할 때 객체의 포함된 정보의 손상과 오용을 막음

데이터가 바뀌여도 다른 객체에 영향을 주지 않아 독립성 유지

**은닉화(hiding)**

캡슐화의 한 개념으로, 내부 데이터와 연산을 외부에서 접근하지 못하도록 은닉하는것

변수에 접근제한자를 private로 지정

**② 추상화(abstraction)**

객체들이 가진 공통의 특성들을 파악하고 불필요한 특성들을 제거하는 과정

즉, 중요한 건 남기고 불필요한 것 & 자세한 것은 생략

추상(抽象) : 여러 가지 사물이나 개념에서 공통되는 특성이나 속성 따위를 추출하여 파악하는 작용.

장점

복잡도 관리

예) 사람 객체라면 – 눈, 코, 이름, 성별, 키, 잔다, 뛴다 등

   ‘날다’라는 추상화는 불가

c++ 코드 예) 동물 객체라면 - 울음소리

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class Animal  {      public:          virtual void Cry()=0; // 순수 가상 함수의 선언  };    class Dog : public Animal  {      public:          virtual void Cry() { cout << "멍멍!!" << endl; }  };  class Cat : public Animal  {      public:          virtual void Cry() { cout << "야옹!!" << endl; }  };  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

Java 코드 예) 계산기 객체라면

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public abstract class Calculator{      int left, right;      public void setOprands(int left, int right){          this.left = left;          this.right = right;      }      public abstract void sum();      public abstract void avg();  }    class CalculatorDecoPlus extends Calculator {      public void sum(){          System.out.println("+ sum :"+(this.left+this.right));      }      public void avg(){          System.out.println("+ avg :"+(this.left+this.right)/2);      }  }    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**③ 상속성(inheritance)**

하나의 클래스(객체)가 다른 클래스(객체)의 특성을 이어받을 수 있게 해주는 과정

자식 클래스는 부모 클래스의 모든 특성을 상속 받고, 자기 자신의 특성을 추가 시켜 정의할 수 있다

장점

코드 중복 작업 최소화로 개발 시간 절약

소프트웨어의 재사용성 증대

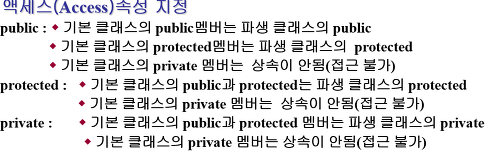
유지 보수 시간 최소화

클래스 상속의 형식

**class** **derived\_class\_name** : **access base\_class\_name**{

}

**access** 속성에 따른 차이점



Ex) 위에서 예시로 쓴 student class에서 pen class가 public으로 상속된다면

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class pen : public student{      private:          int num;      public:          pen(int a, int b) : student(a){              num = b;          }  }  int main(){   pen p(10,20);   p.show(); // 10출력     return 0;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

student 클래스의 show함수를 pen 객체가 사용가능

**④ 다형성(polymorphism)**

목적은 다르지만 연관성 있는 두 가지 이상의 용도로 하나의 이름을 사용할 수 있게 하는 성질

장점

객체의 부분화 가능

함수 재정의

⒜ **function overloading**: 같은 이름의 메소드를 여러 개 가지면서 매개변수의 유형과 개수가 다르게 하는 기술

(생성자 오버로딩 같은 거를 말함)

⒝ **function overrriding**: 상위클래스가 가지고 있는 메소드를 하위 클래스가 재정의 하는 기술

   위에서 상속 예시를 든 pen class로 알아보자면

class pen: public student{

           private:

                      int num;

           public:

                      void show(){

           cout << score << num;

}

}

   부모 클래스의 **show()함수**를 자식 클래스인 pen에서 재정의

