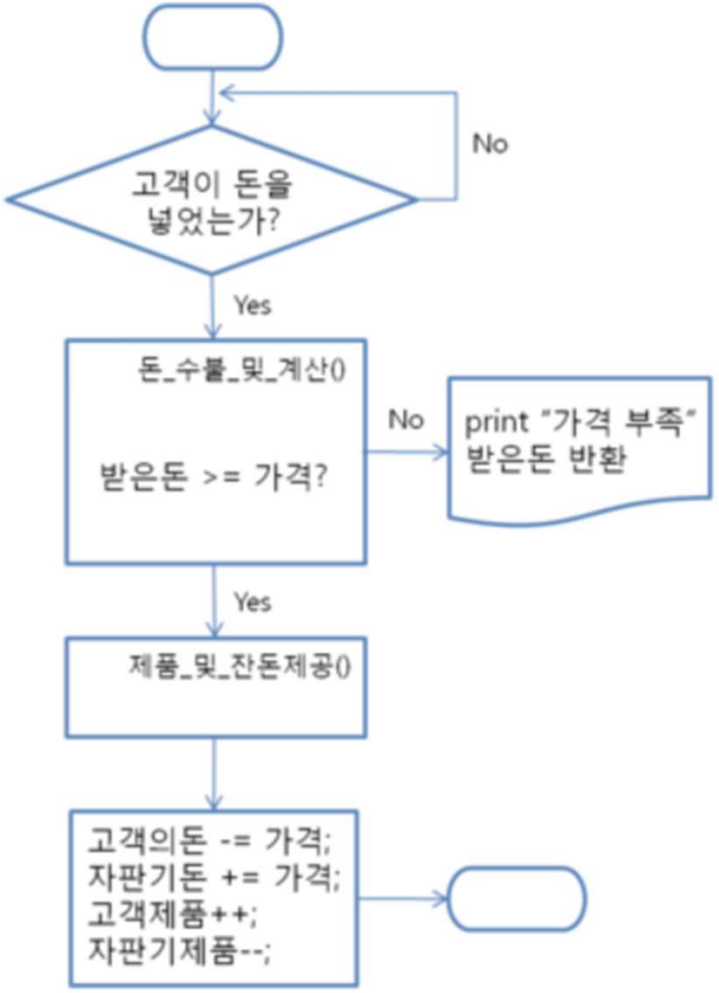
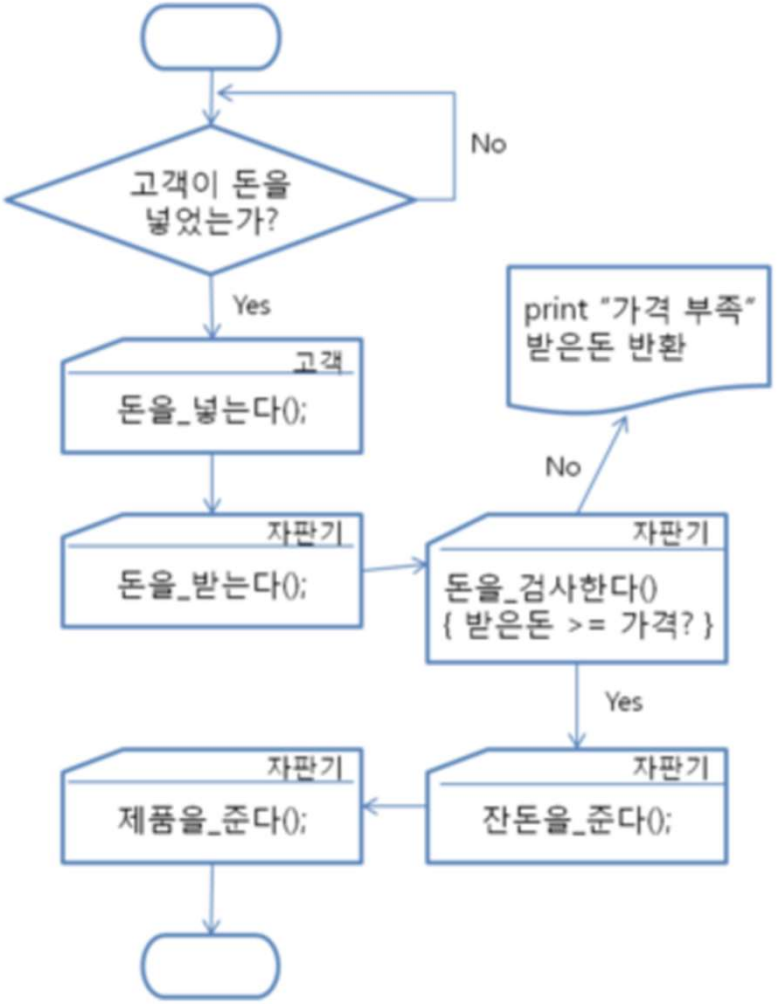

CODINGON
X
posco
K-Digital Training
스마트 팩토리 4기

클래스

절차지향 방식



객체지향 방식



객체 지향 프로그래밍의 4가지 특징

추상화

캡슐화

상속

다형성

추상화(Abstraction)

[실생활]
사물의 공통성, 본질을 모아 추출하여 파악하는 것.

[프로그래밍]
객체의 공통적인 속성과 기능을 추출하여 정의하는 것.
속성은 클래스 내에 필드(변수)로 정의 될 것이고,
기능은 클래스 내에서 메소드(함수)로 정의될 것이다.

캡슐화(Encapsulation)

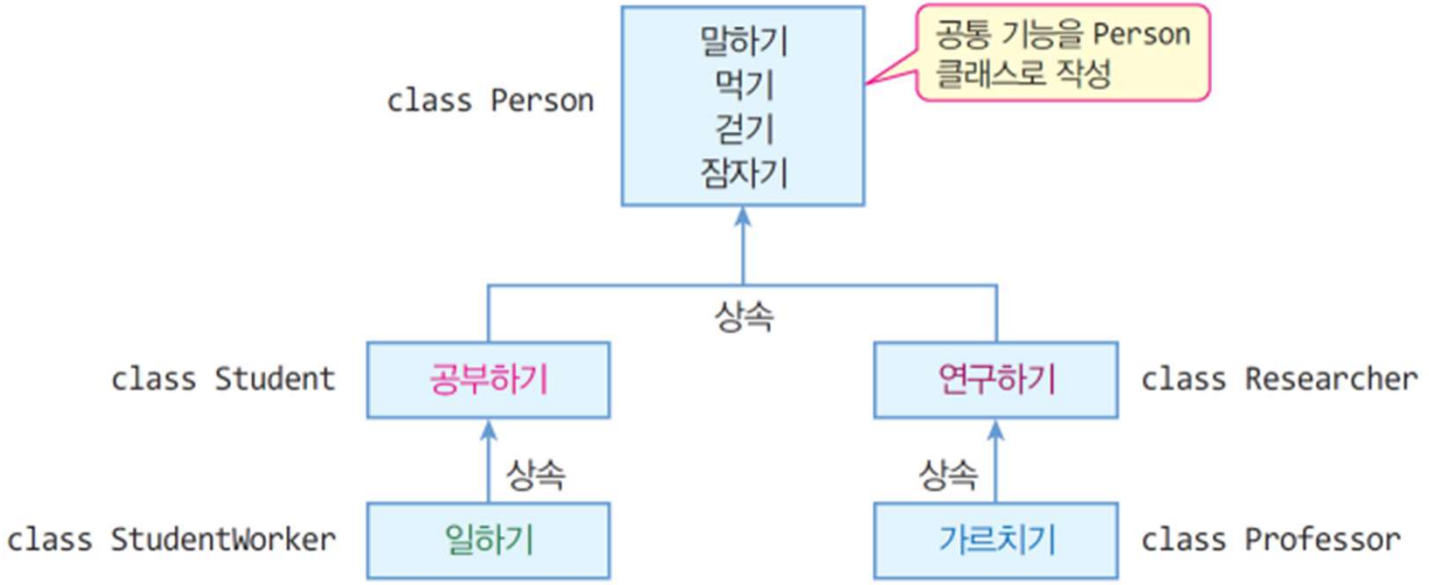
연관 있는 속성과 기능들을 하나의 캡슐로 만들어 내부의 데이터들을 외부로부터 보호하는 것.

- 데이터 보호
  외부로부터 클래스에 정의된 속성과 기능들을 보호

- 데이터 은닉
  내부의 동작을 감추고 외부에는 필요한 부분만 노출

상속(Inheritance)

부모 클래스에 정의된 속성(변수) 및 기능(메서드)들을 자식 클래스에서 상속받아 사용하는 것



다형성(Polymorphism)

같은 대상이라도 문맥이나 상황에 따라 다르게 사용될 수 있다는 원리

[실생활]
내일 배를 타고 제주도에 갈 것이다.
길동이는 배가 아파서 학교에 가지 못했다.
어머니께서 맛있는 배를 깎아 주셨다.
⇒ 같은 글자인데도 문맥에 따라 서로 다르게 해석됨

다형성

[실생활]

스케치북에 도형을 그렸다.

⇒ 여기서 도형은 삼각형이 될 수도, 사각형이 될 수도, 원이 될 수도 있음.

⇒ 도형과 같이 넓은 범위의 객체는 작은 범위의 것들로 대체 될 수 있음.

[프로그래밍]

상위(부모) 클래스로 하위(자식) 클래스의 인스턴스를 생성할 수 있음.

- 오버라이딩 : 부모 클래스 메서드를 자식 클래스에서 재정의
- 오버로딩 : 한 클래스에서 메소드 이름은 같지만 파라미터 개수나 자료형을 다르게 하여 서로 다르게 동작하게 하는 것

클래스 상속

상속 문법

```
class Person {


}
class Student : public Person {
// 여기서 Person 클래스를 Student 클래스가 상속 받음
}
```

Person 클래스에 있는 멤버(변수, 메소드)를 Student 클래스에서 사용할 수 있게 됨.

접근제어자 - protected

```
class Person {
    protected:
        string name;
        int age;
}
class Student : public Person {
    public:
        void test() { name = "홍길동"; }
}
```

접근제어자

```
class Person {
    protected:
        string name;
        int age;
}
class Student : public Person {
    public:
        void test() { name = "홍길동"; }
}
```

??

접근제어자

```
class Person {
    protected:
        string name;
        int age;
}
class Student : public Person {
    public:
        void test() { name = "홍길동"; }
}
```

부모로부터 상속받은 멤버들의 접근제어자 한계를 지정함.

접근제어자

```
class Base {
public:
    int publicVar = 10;

    void publicFunc() {
        cout << "Base::publicFunc()" << endl;
    }

    Base() {
        cout << "Base constructor" << endl;
    }
};

class Derived : private Base {
public:
    void accessBaseMembers() {
        // Within the derived class, you can access base members directly.
        cout << publicVar << endl;    // OK
        publicFunc();                 // OK
    }
};

int main() {
    Derived d;
    d.accessBaseMembers();  // This is fine because it's called from within the derived class.

    // Outside the derived class, you cannot access base members directly:
    // d.publicVar;     // Error: 'Base' is an inaccessible base of 'Derived'
    // d.publicFunc();  // Error: 'Base' is an inaccessible base of 'Derived'

    return 0;
}
```

실습1 상속 사용해보기

(1) Shape라는 클래스를 만들어주세요. 이 클래스는 아래 조건을 만족해야 합니다.

조건 1. 변의 개수, 밑변의 길이를 저장하는 변수를 가지고 있어야 합니다

조건 2. 변의 개수와 밑변의 길이를 출력하는 printInfo() 함수를 가지고 있어야 합니다.

(2) Shape 클래스를 상속 받는 Rectangle, Triangle 클래스를 만들어주세요. 이 클래스들은 아래 조건을 만족해야 합니다.

조건 1. Rectangle 클래스에는 세로 길이를 의미하는 변수를 가지고 있어야 합니다.

조건 2. Triangle 클래스에는 높이 길이를 의미하는 변수를 가지고 있어야 합니다.

조건 3. 두 클래스에는 각각 도형의 넓이를 구하고 출력하는 area() 함수를 가지고 있어야 합니다.

조건 4. 두 클래스 모두 생성자에서 모든 변수에 값을 대입해주세요. (변, 밑변도 대입)

(3) 메인 함수에서 Triangle과 Rectangle 클래스를 통해 각각 인스턴스를 만들고 area() 함수를 실행시키도록 만들어주세요.

오버라이딩

오버라이딩 ??

- 상속 관계에 있는 부모 클래스에서 이미 정의된 메소드를 자식 클래스에서 다시 정의하는 것

- 매개변수의 유형과 개수가 같은 완전히 같은 메소드를 재정의 하는 것!

오버라이딩

```
class Person {
public:
    void sleep() { cout << "잠자기" << endl; }
};

class Student : public Person {
public:
    void sleep() { cout << "잠자기 오버라이딩" << endl; }
};
```

실습2 오버라이딩 이해하기

(1)실습1에서 구현한 printInfo() 오버라이딩.

-> Rectangle 클래스에서는
"사각형의 넓이는 {넓이}" 출력

-> Triangle 클래스에서는
"삼각형의 넓이는 {넓이}" 출력

오버로딩

오버로딩 ??

- 같은 이름의 메소드를 중복하여 정의하는 것을 의미함.

- 매개변수의 유형이나 개수가 달라야 함

실습3 오버로딩 이해하기

(1)실습1에서 구현한 printInfo() 오버로딩.

(2)printInfo(int w, int h)
-> 밑변의 길이와 높이를 입력받아서 넓이 출력하는 함수 구현