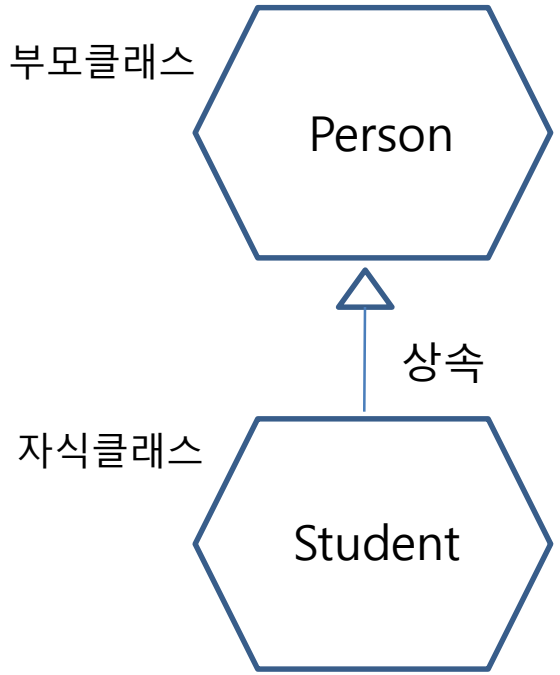
C++_상속, 다형성

Visual Studio 2022

상속(Inheritance)

- 상속이란?

  기존에 있던 클래스를 이용해서 새로운 클래스를 만드는 것이다.
  이때 상속을 해준 클래스를 부모클래스 또는 수퍼클래스라 하고, 상속받은 새로운 클래스를 자식 클래스 또는 서브클래스라 한다.




```
class 클래스이름 : 부모클래스 이름{
    멤버 리스트
}
```

```
class Person{
    멤버 리스트
};
class Student : public Person{
    멤버 리스트
};
```

콜론(:) 1개 사용
public 사용

상속의 선언과 활용

- 부모 클래스 정의

```
//Person 클래스 정의
class Person {
private:
    string name;
    int age;
```

```
public:
    //기본 생성자 생략됨
    void setName(string name) {
        this->name = name;
    }

    string getName() {
        return name;
    }

    void setAge(int a) {
        age = a;
    }

    int getAge() {
        return age;
    }
};
```

상속의 선언과 활용

- 자식 클래스 정의 : 상속

```
//Person을 상속한 Student 클래스 정의
class Student : public Person {
private:
    int studentId;

public:
    void setStudentId(int id) {
        studentId = id;
    }

    int getStudentId() {
        return studentId;
    }
};
```

상속의 선언과 활용

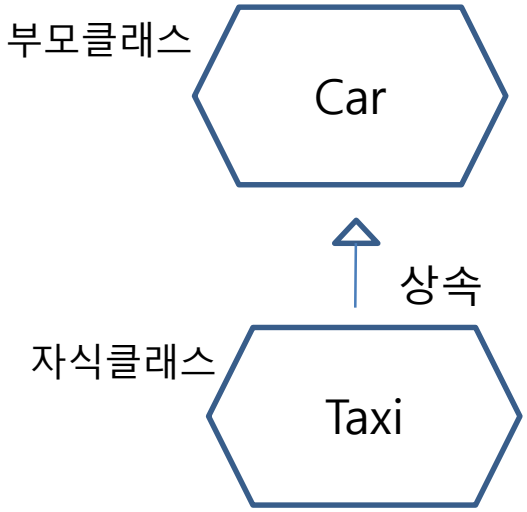
- 상속 테스트

```
//부모 객체 생성
Person p1;
p1.setName("광개토태왕");
p1.setAge(40);

//사람의 정보
cout << "이름: " << p1.getName() << endl;
cout << "나이: " << p1.getAge() << endl;

//자식 객체 생성
Student s1;
s1.setName("장수왕");
s1.setAge(97);
s1.setStudentId(1000);

//학생의 정보
cout << "이름: " << s1.getName() << endl;
cout << "나이: " << s1.getAge() << endl;
cout << "학번: " << s1.getStudentId() << endl;
```

```
이름: 광개토태왕
나이: 40
이름: 장수왕
나이: 97
학번: 1000
```

매개변수가 있는 생성자 상속

- 매개변수가 있는 생성자 상속




```
class Car{
    멤버 리스트
};
Class Taxi : public Car{
    멤버 리스트
};
```

생성자

```
Taxi(int passenger, string serial, int speed) : Car(serial, speed) {
    this->passenger = passenger;
}
```

매개변수가 있는 생성자 상속

- 부모 클래스 정의

```
class Car {
private:
    string serial;
    int speed;

public:
    /*Car(string serial, int speed) {
        this->serial = serial;
        this->speed = speed;
    }*/

    Car(string serial, int speed) : serial(serial), speed(speed) {}

    string getSerial() {return serial;}
    int getSpeed() {return speed;}
    void carInfo() {
        cout << "차량 번호: " << getSerial() << endl;
        cout << "주행 속도: " << getSpeed() << endl;
    }
};
```

매개변수가 있는 생성자 상속

- 자식 클래스 정의 : 상속

```
class Taxi : public Car {
private:
    int passenger;

public:
    /*Taxi(int passenger, string serial, int speed) : Car(serial, speed) {
        this->passenger = passenger;
    }*/

    Taxi(int passenger, string serial, int speed) :
        Car(serial, speed) , passenger(passenger) {}

    int getPassenger() {return passenger;}
    void carInfo() {  //부모 함수 재정의(Overriding)
        cout << "차량 번호: " << getSerial() << endl;
        cout << "주행 속도: " << getSpeed() << endl;
        cout << "승객수: " << getPassenger() << endl;
    }
};
```

함수 재정의 (Overriding)

매개변수가 있는 생성자 상속

- 상속 테스트

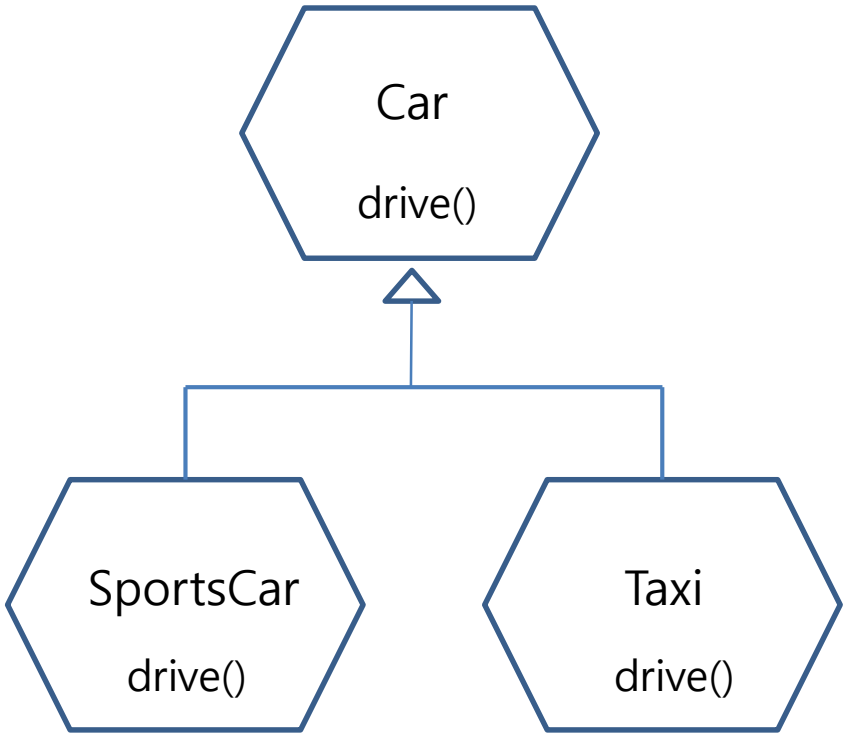
```
int main()
{
    //Car 객체 생성
    Car car1("02허 2424", 80);
    car1.carInfo();
    cout << "========================\n";

    //Taxi 객체 생성
    Taxi 카카오T(2, "123가 4567", 60);
    카카오T.carInfo();

    return 0;
}
```

```
차량 번호: 02허 2424
주행 속도: 80
========================
차량 번호: 123가 4567
주행 속도: 60
승객수: 2
```

멤버 함수 재정의 – 오버라이딩

- 오버라이드(Override)

  부모 클래스의 멤버 함수를 다시 자식 클래스에서 정의하는 것으로 함수 재정의(Override)라 한다.



함수 재정의(Overriding)

- 오버라이딩(Overriding)

```
//함수 재정의
class Car {
public:
    void drive() {
        cout << "차가 달립니다." << endl;
    }
};

class Taxi : public Car {
public:
    void drive() {
        cout << "택시가 달립니다." << endl;
    }
};

class SportsCar : public Car {
public:
    void drive() {
        cout << "스포츠카가 달립니다." << endl;
    }
};
```

함수 재정의(Overriding)

- 오버라이딩(Overriding)

```
int main()
{
    Car car;
    car.drive();

    SportsCar sCar;
    sCar.drive();

    Taxi taxi;
    taxi.drive();


    return 0;
}
```

```
차가 달립니다.
스포츠카가 달립니다.
택시가 달립니다.
```

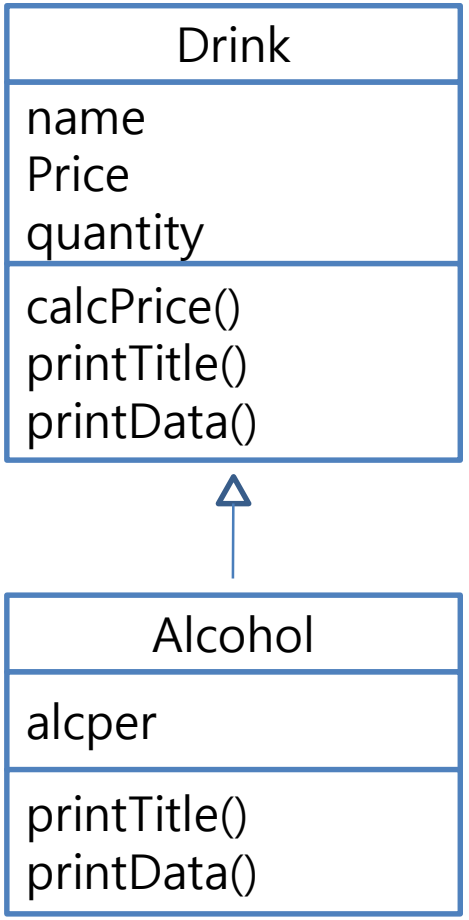
상속 – Protected 접근지정자

- 접근 지정자

접근 지정자	설 명
public	외부 클래스 어디에서나 접근 할수 있다.
protected	**클래스 내부와 상속관계의 모든 자식 클레스에서 접근 가능**
private	같은 클래스 내부 가능, 그 외 접근 불가

상속 – Protected 접근지정자

- 매출 전표 작성하기




```
============= 매출 전표 =============
상품명   가격     수량     금액
커피     2500     4        10000
녹차     3000     3        9000

상품명(도수[%]) 가격     수량     금액
soju(15.1)      4000     2        8000

*** 합계 금액: 27000원 ***
```

상속 – Protected 접근지정자

- 매출 전표 작성하기

```
class Drink {
protected:
    string name;    //상품명
    int price;      //가격
    int quantity;   //수량

public:
    /*Drink(string name, int price, int quantity) {
        this->name = name;
        this->price = price;
        this->quantity = quantity;
    }*/

    Drink(string name, int price, int quantity) :
        name(name), price(price), quantity(quantity){ }

    int calcPrice() { return price * quantity;};
    static void printTitle() { cout << "상품명\t가격\t수량\t금액\n";}
    void printData() {
        cout << name << "\t" << price << "\t" << quantity <<
                    "\t" << calcPrice() << endl;
    }
};
```

상속 – Protected 접근지정자

- 매출 전표 작성하기

```
class Alcohol : public Drink{
private:
    float alcper;  //알콜 도수

public:
    /*Alcohol(float alcper, string name, int price, int quantity) : Drink(name, price, quantity) {
        this->alcper = alcper;
    }*/

    Alcohol(float alcper, string name, int price, int quantity) :
        Drink(name, price, quantity), alcper(alcper) {}

    static void printTitle() { //함수 재정의
        cout << "상품명(도수[%])\t가격\t수량\t금액\n";
    }

    void printData() {
        cout << name << "(" << alcper << ")\t" << price << "\t" <<
            quantity << "\t" << calcPrice() << endl;
    }
};
```

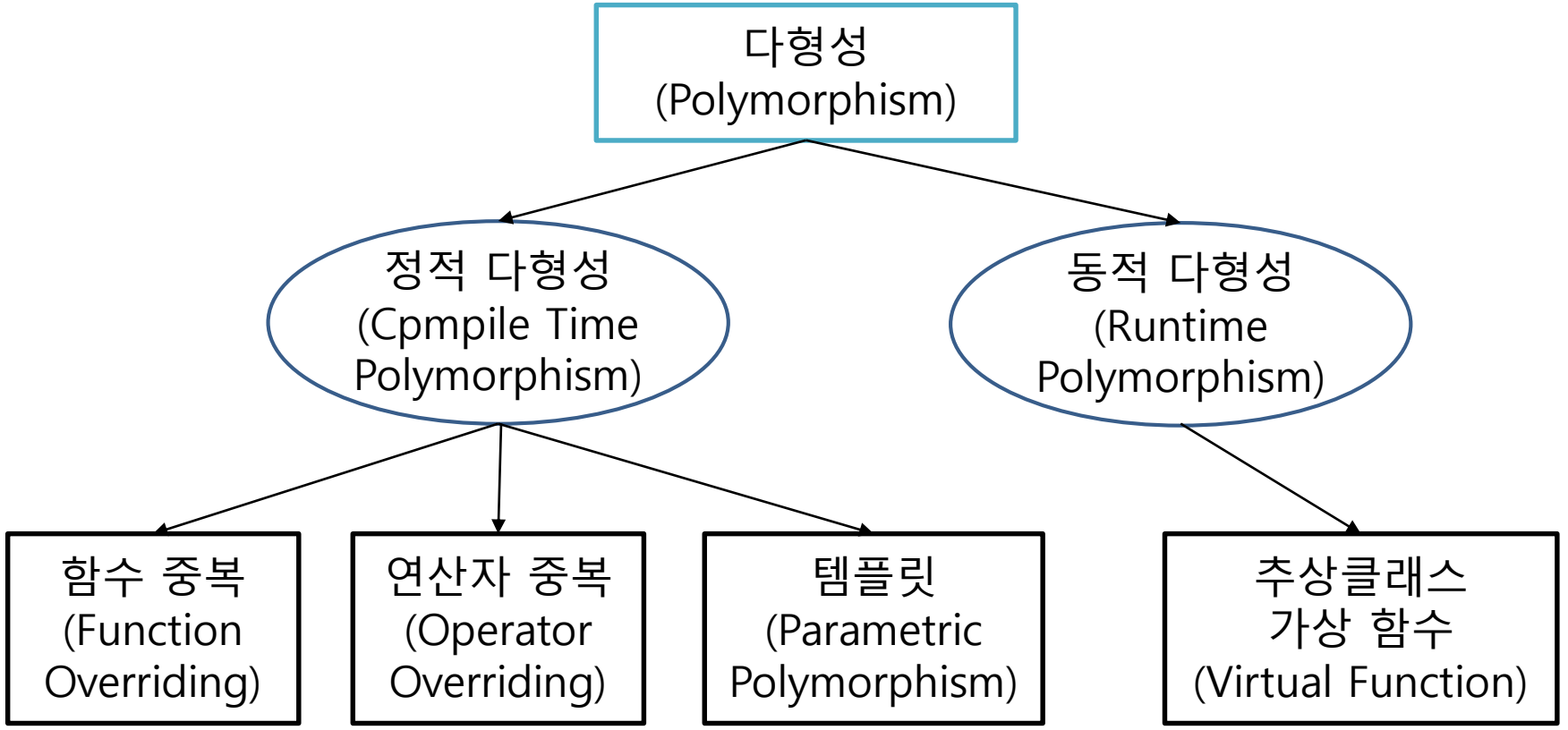
상속 – Protected 접근지정자

- 매출 전표 작성하기

```
int main()
{
    Drink coffee("커피", 2500, 4);
    Drink tea("녹차", 3000, 3);

    cout << "============= 매출 전표 =============\n";
    Drink::printTitle(); //클래스 이름으로 직접 접근
    coffee.printData();
    tea.printData();
    cout << endl;

    Alcohol soju(15.1f, "soju", 4000, 2);
    Alcohol::printTitle();
    soju.printData();

    int total;
    total = coffee.calcPrice() + tea.calcPrice() + soju.calcPrice();
    cout << "***** 합계 금액: " << total << "원 *****\n";

    return 0;
}
```

다형성(Polymorphism)

다형성이란? 다양한 종류의 객체에게 동일한 메시지를 보내더라도 각 객체들이 서로 다르게 동작하는 특성을 말한다.



C 언어에 추가한 기능

- C++ 언어는 C언어의 문법적 규칙을 그대로 승계
  - **함수 중복(function overloading)** – 매개 변수의 개수나 타입이 서로 다른 동일한 이름의 함수들을 선언할 수 있게 한다.
  - **참조(reference)와 참조 변수** – 변수에 별명을 붙여 변수 공간을 같이 사용할 수 있다.
  - **new와 delete 연산자** – 동적 메모리 할당, 해제를 위한 new, delete연산자를 도입
  - **연산자 재정의(operator overloading)** – 기존의 연산자에 새로운 연산을 정의할 수 있게 한다.
  - **클래스와 제네릭 함수(generics)** – 함수나 클래스를 데이터 타입에 의존하지 않고 일반화 시킬수 있게 한다.

가상함수와 동적 결합(Dynamic Binding)

- 다형성에 의해 함수 재정의시 요구 조건
  - 부모 클래스의 멤버 함수가 가상함수(추상함수)로 선언되어야 함
  - **virtual** 키워드를 사용한다.
  - 함수 구현부의 내용은 비워둔다.

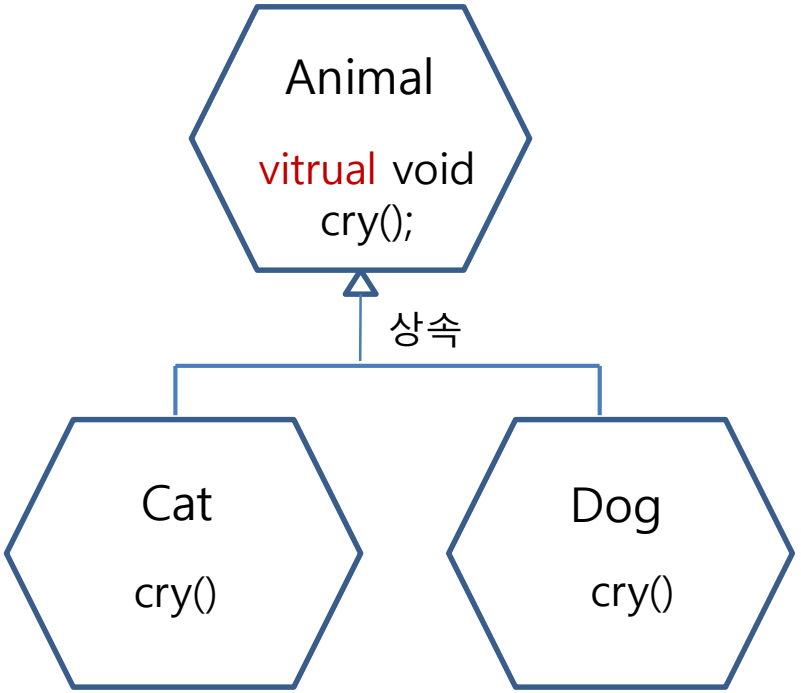
```
virtual cry() { }
```

- 동적 결합 (Dynamic Binding)
  - 실행시 호출될 함수를 결정하는 것으로 이는 하나의 함수가 여러 클래스에서 오버라이딩 되었을 때 사용한다.
  - 객체 생성시 **new**, 해제 시 **delete** 사용

```
Animal* cat = new Cat     부모클래스 = new 자식클래스(자동 형변환 )
```

가상(Virtual) 함수

- 가상 함수 사용



가상(Virtual) 함수

- 가상 함수 사용

```
//가상 함수
class Animal {
public:
    void breathe() {
        cout << "숨을 쉽니다." << endl;
    }
    virtual void cry() {};  //가상(추상) 함수
};

class Cat : public Animal {
public:
    void cry() {
        cout << "야옹~" << endl;
    }
};

class Dog : public Animal {
public:
    void cry() {
        cout << "멍~ 멍~" << endl;
    }
};
```

가상(Virtual) 함수

- 가상 함수 사용

```
int main()
{
    //정적 객체 생성
    /*Cat cat;
    cat.breathe();
    cat.cry();*/

    //동적 객체 생성
    Animal* cat = new Cat;
    Animal* dog = new Dog;

    cat->breathe();
    cat->cry();

    dog->breathe();
    dog->cry();

    delete cat;  //메모리 해제
    delete dog;

    return 0;
}
```

```
숨을 쉽니다.
야옹~
숨을 쉽니다.
멍~ 멍~
```

연산자 오버로딩(중복)

- 연산자 오버로딩
  - ✓ 연산자를 재정의하여 사용자 정의 클래스로 사용하는 것을 말한다.

함수반환형 **Operator** 연산자 (연산대상){ … }

연산자 오버로딩(중복)

- 객체 더하기

```
//연산자 오버로딩
class Point {
private:
    int x, y;
public:
    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }

    void print() {
        cout << "x=" << x << ", y=" << y << endl;
    }

    //더하기 연산 함수
    Point operator+(Point p) {
        x = x + p.x;
        y = y + p.y;
        return Point(x, y);
    }
};
```

연산자 오버로딩(중복)

- 객체 더하기

```
int main()
{
    //점 객체 생성
    Point p1(1, 2);
    Point p2(3, 4);

    p1.print();
    p2.print();

    //객체 더하기
    Point p3 = p1 + p2;

    p3.print();

    return 0;
}
```

```
x=1, y=2
x=3, y=4
x=4, y=6
```

연산자 오버로딩(중복)

- 객체의 크기 비교(비교 연산)

```
class Circle {
    double radius;
public:
    Circle(double radius) {
        this->radius = radius;
    }
    double getRadius() { return radius; }
    double getArea() { return PI * radius*radius; }
    bool operator >= (Circle c);
};

bool Circle::operator>=(Circle c) {
    if (this->radius >= c.radius)
        return true;
    else
        return false;
}
```

연산자 오버로딩(중복)

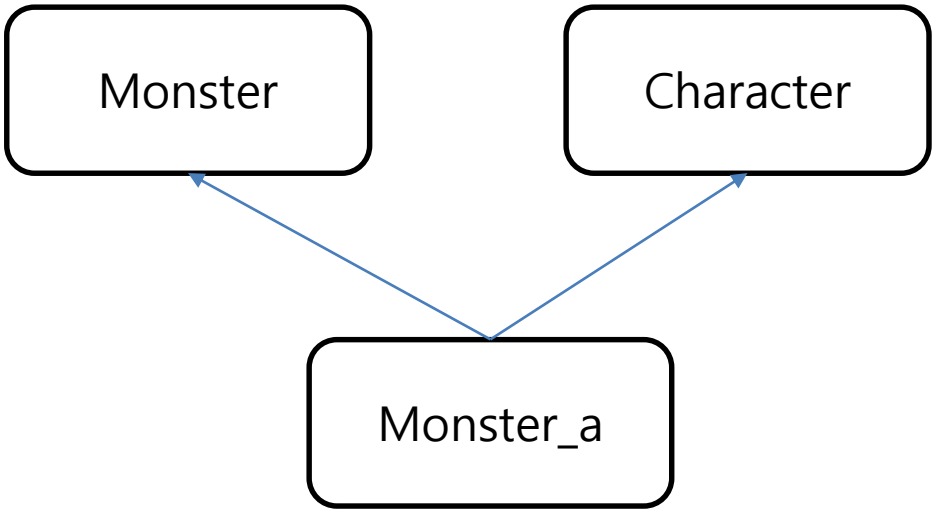
- 객체의 크기 비교(비교 연산)

```
int main()
{
    Circle c1(5.1), c2(12.3);
    cout << "원1의 반지름 : " << c1.getRadius() << endl;
    cout << "원1의 면적 : " << c1.getArea() << endl;
    cout << "원2의 반지름 : " << c2.getRadius() << endl;
    cout << "원2의 면적 : " << c2.getArea() << endl;

    if (c1 >= c2)
        cout << "객체 c1이 c2보다 크다." << endl;
    else
        cout << "객체 c2가 c1보다 크다." << endl;
    return 0;
}
```

다중 상속(Multiple Inheritance)

- 다중상속(multiple inheritance)

  하나의 파생 클래스가 여러 클래스를 동시에 상속받는 것이다.



다중 상속(Multiple Inheritance)

- Character, Monster 클래스

```
class Character {
public:
    Character() {
        cout << "Character 클래스 생성자" << endl;
    }
    ~Character() {
        cout << "Character 클래스 소멸자" << endl;
    }
};

class Monster {
public:
    Monster() {
        cout << "Monster 클래스 생성자" << endl;
    }
    ~Monster() {
        cout << "Monster 클래스 소멸자" << endl;
    }
};
```

다중 상속(Multiple Inheritance)

- Character, Monster 클래스를 상속받은 MonsterA 클래스

```
class MonsterA : public Monster, Character {
private:
    int location[2]; //좌표 저장

public:
    //기본생성자 : 초기화 목록
    MonsterA() : MonsterA(0, 0) {
        cout << "MonsterA 클래스 생성자" << endl;
        //MonsterA(0, 0); //초기화 되지 않음
    }

    MonsterA(int x, int y) : location{ x, y } {
        cout << "MonsterA 클래스 생성자(매개변수 추가)" << endl;
    }

    void showLocation() {
        cout << "위치(" << location[0] << ", " << location[1] << ")" << endl;
    }
};
```

다중 상속(Multiple Inheritance)

- Character, Monster 클래스를 상속받은 MonsterA 클래스

```
int main()
{
    MonsterA forestMonster;   //기본 생성자 호출
    forestMonster.showLocation();

    MonsterA woodMonster(10, 20); //매개변수가 있는 생성자 호출
    woodMonster.showLocation();

    return 0;
}
```

```
Monster 클래스 생성자
Character 클래스 생성자
MonsterA 클래스 생성자(매개변수 추가)
MonsterA 클래스 생성자
위치(0, 0)
Monster 클래스 생성자
Character 클래스 생성자
MonsterA 클래스 생성자(매개변수 추가)
위치(10, 20)
Character 클래스 소멸자
Monster 클래스 소멸자
Character 클래스 소멸자
Monster 클래스 소멸자
```