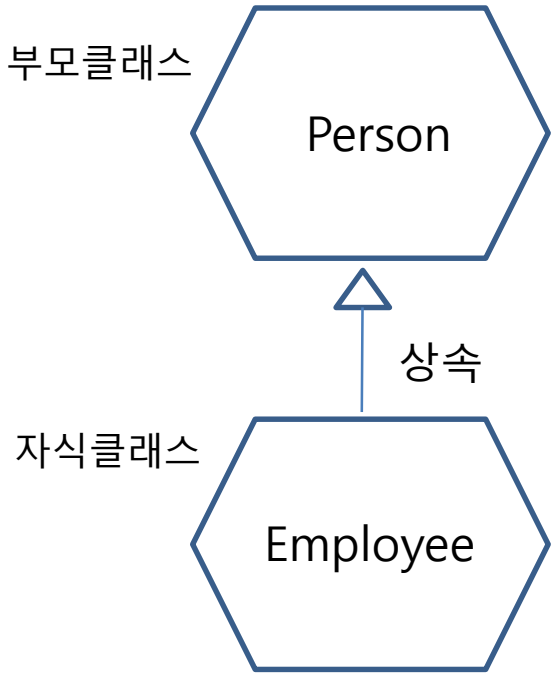
7장. 상속과 다형성

Visualstudio 2019

상속(Inheritance)

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




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

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

콜론(:) 1개 사용
public 사용

상속(Inheritance)

- Person을 상속한 Employee 클래스

employee.h

```
class Person {
public:
    string name;
    int age;

    Person(string name, int age = 0);
};

Person::Person(string name, int age) : name(name), age(age){}
```

```
class Employee : public Person {
public:
    int companyID;

    Employee(int companyID = 0, string name = "unnamed", int age = 0);
};

Employee::Employee(int companyID, string name, int age)
    : companyID(companyID), Person(name, age) {}
```

상속(Inheritance)

- Person을 상속한 Employee 클래스

```
Employee e1 = Employee();
Employee e2 = Employee(1002);
Employee e3 = Employee(1003, "삼구", 22);

e1.name = "일구";
e1.age = 28;
e1.companyID = 1001;

e2.name = "이구";
e2.age = 31;

cout << e1.name << "의 사번은 " << e1.companyID << "입니다." << endl;
cout << e2.name << "의 사번은 " << e2.companyID << "입니다." << endl;
cout << e3.name << "의 사번은 " << e3.companyID << "입니다." << endl;
```

상속 – protected 접근지정자

- protected 접근 지정자 사용

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

calculator.cpp

```
class Calculator {
public:
    int add(int x, int y) { return x + y;}

protected:
    int sub(int x, int y) { return x - y; }

private:
    int mul(int x, int y) { return x * y; }
};
```

상속 – Protected 접근지정자

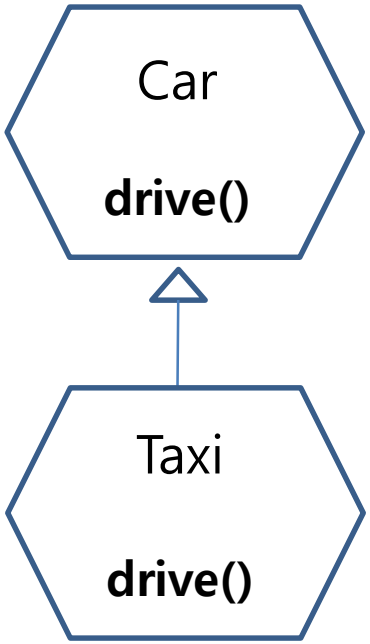
- protected 접근 지정자 사용

```
class MyCalculator : public Calculator {
public:
    double div(int x, int y) { return x /(double) y; }

    void access() {
        cout << "add(10, 4) : " << add(10, 4) << endl;
        cout << "sub(10, 4) : " << sub(10, 4) << endl;
        //cout << "mul(10, 4) : " << mul(10, 4) << endl; //접근 불가
        cout << "div(10, 4) : " << div(10, 4) << endl;
    }
};

int main() {
    MyCalculator calc = MyCalculator();
    calc.access();

    return 0;
}
```

상속 – 함수 재정의(오버라이딩)

- 함수 재정의(overriding)

  부모 클래스의 멤버 함수와 이름은 같지만 다르게 재정의 하는 것



car.h

```
class Car {
    string model;
    int speed;

public:
    Car(string model, int speed);
    string getModel();
    int getSpeed();
    void drive();
};

class Taxi : public Car {
    int passenger;

public:
    Taxi(int passenger, string model, int speed);
    int getPassenger();
    void drive();
};
```

상속 – Protected 접근지정자

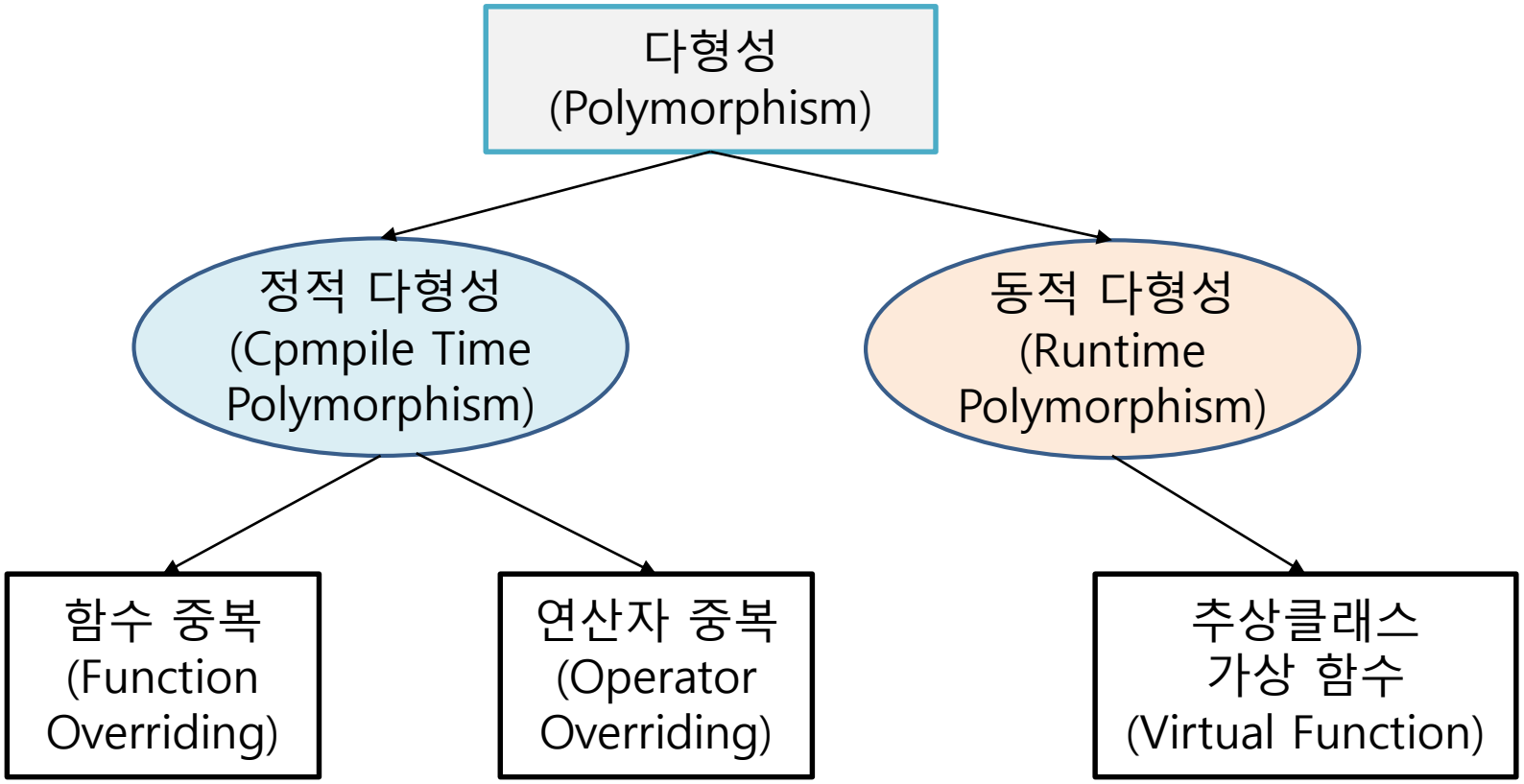
```
#include "car.h"

Car::Car(string model, int speed) {
    this->model = model;
    this->speed = speed;
}

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

string Car::getModel() { return model;}

int Car::getSpeed() { return speed; }

int Taxi::getPassenger() { return passenger; }

void Car::drive() { cout << "차가 달립니다." << endl; }

void Taxi::drive() { cout << "택시가 달립니다." << endl; } //함수 재정의
```

상속 – Protected 접근지정자

```
#include "car.h"

int main() {

    Taxi taxi = Taxi(3, "Sonata", 60);

    cout << taxi.getModel() << " 택시의 승객은 " << taxi.getPassenger()
        << "명입니다. " << endl;
    cout << taxi.getSpeed() << "KM 속도로 ";
    taxi.drive();

    return 0;
}
```

다형성(Polymorphism)

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



가상함수와 동적 결합(Dynamic Binding)

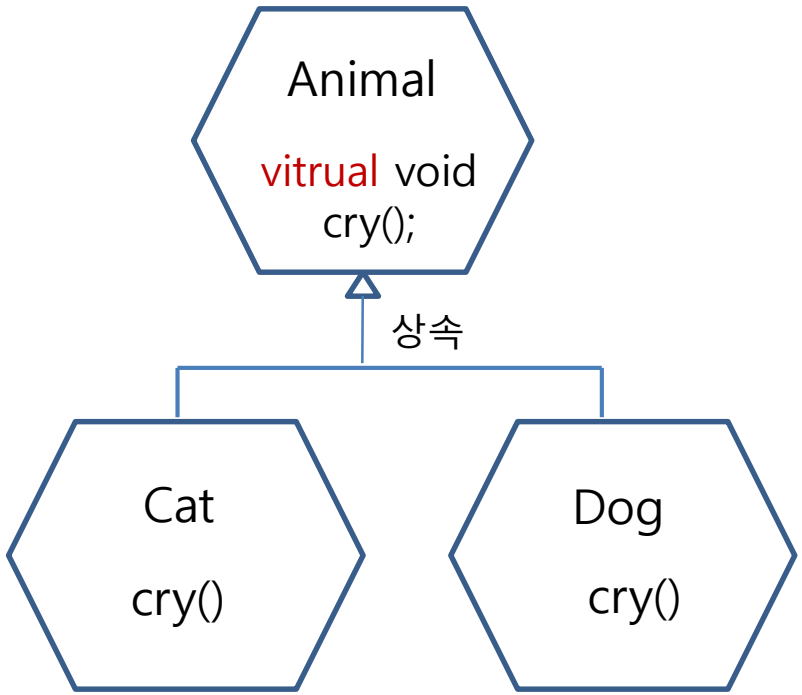
- 가상함수
  - 부모 클래스의 멤버 함수가 가상함수(추상함수)로 선언되어야 함
  - **virtual** 키워드를 사용한다.

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

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

```
Animal* cat = new Cat

부모클래스 = new 자식클래스(자동 형변환 )
```

가상함수와 동적 바인딩



animal.h

```
class Animal {
public:
    void breathe() {
        cout << "동물은 숨을 쉰다." << endl;
    }
    virtual void cry() {};
};

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

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

가상함수와 동적 바인딩

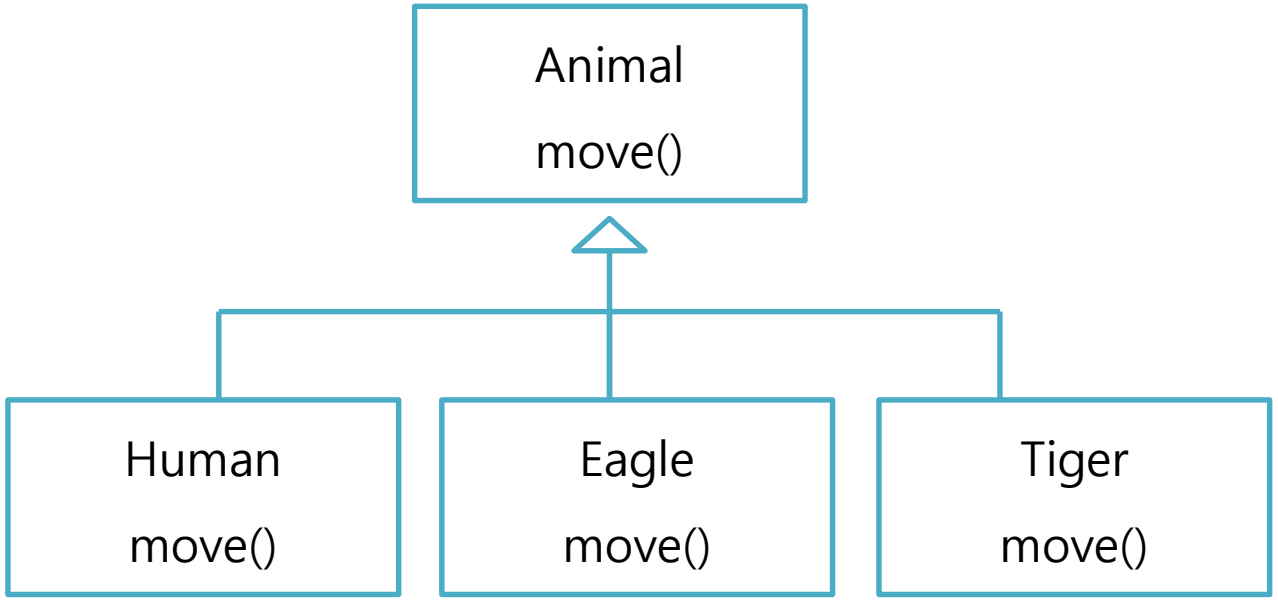
animal_main.cpp

```
//Cat cat = Cat();
Animal* cat = new Cat();
//동적 객체 생성 - 다형성(부모 클래스로 객체 생성)
cat->breathe();
cat->cry();

Animal* dog = new Dog();
dog->breathe();
dog->cry();

delete cat;
delete dog;
```

다형성(polymorphism)

- **다형성이란?**

  하나의 타입(자료형)에 대입되는 객체에 따라서 실행결과가 다양한 형태로 나오는 성질을 말한다..



다형성(polymorphism) – 매개변수

- 매개변수의 다형성

  매개값을 다양화하기 위해 매개변수를 부모타입으로 선언하고 호출할때 자식객체를 대입.

```
class Animal {
public:
    virtual void move(); //가상함수 선언
};

class Human : public Animal {
public:
    void move();
};

class Eagle : public Animal {
public:
    void move();
};

class Tiger : public Animal {
public:
    void move();
};
```

```
#include "animal.h"

void Animal::move() {
    cout << "동물이 움직입니다." << endl;
}

void Human::move() {
    cout << "사람이 두 발로 걷습니다." << endl;
}

void Eagle::move() {
    cout << "독수리가 하늘을 납니다." << endl;
}

void Tiger::move() {
    cout << "호랑이가 네 발로 달립니다." << endl;
}

void moveAnimal(Animal* animal) {
    animal->move();
}
```

다형성(polymorphism) – 매개변수

```
#include "animal.h"

void moveAnimal(Animal* animal);

int main() {

    Animal* human = new Human();
    Animal* eagle = new Eagle();
    Animal* tiger = new Tiger();

    moveAnimal(human);
    moveAnimal(eagle);
    moveAnimal(tiger);

    delete human;
    delete eagle;
    delete tiger;

    return 0;
}
```

매개변수의 다형성

연산자 오버로딩(중복)

- 연산자를 재정의하여 사용자 정의 클래스로 사용하는 것을 말한다.

  [함수반환형 **Operator** 연산자 (연산대상)]

```
class Point {
    int x, y;

public:
    Point(int x = 0, int y = 0);
    Point operator+(Point p);
    void print();

};
```

```
Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}

Point Point::operator+(Point p) {
    x = x + p.x;
    y = y + p.y;
    return Point(x, y);
}

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

연산자 오버로딩(중복)

- 연산자 오버로딩

```
#include "point.h"

int main() {
    Point p1 = Point(1, 2);
    Point p2 = Point(3, 4);
    Point p3 = p1 + p2;

    p3.print();

    return 0;
}
```

연산자 오버로딩(중복)

- 객체의 크기 비교
  (비교 연산)

```
class Circle {
    double radius;
public:
    Circle(double radius = 0.0);
    double getRadius();
    double getArea();
    bool operator>=(Circle c);
};

Circle::Circle(double radius) : radius(radius) {}

double Circle::getRadius() { return radius; }

double Circle::getArea() { return PI * radius * radius; }

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

연산자 오버로딩(중복)

- 객체의 크기 비교(비교 연산)

```
Circle c1 = Circle(4.2);
Circle c2 = Circle(11.5);

cout << "원1의 반지름 : " << c1.getRadius() << endl;
cout << "원2의 반지름 : " << c2.getRadius() << endl;
cout << "원1의 면적 : " << c1.getArea() << endl;
cout << "원2의 면적 : " << c2.getArea() << endl;

if (c1 >= c2)
    cout << "객체 c1이 c2보다 크다" << endl;
else
    cout << "객체 c1이 c2보다 작다" << endl;
```

예외 처리(Exception Handling)

프로그램 실행 중에 예외가 발생한 경우에 대한 처리 과정

예외처리가 없는 경우 정상적인 프로그램 실행이 안 될 수도 있다.

0으로 나누었을때 예외 처리

```
int n1, n2;
int quotient, reminder;

cout << "수1 : "; cin >> n1;
cout << "수2 : "; cin >> n2;

//if (n2 == 0)
        //cout << n1 << "은 0으로 나눌 수 없습니다!" << endl;

quotient = n1 /(double)n2;
reminder = n1 % n2;
cout << "몫 : " << quotient << endl;
cout << "나머지 : " << reminder << endl;
```

try~catch 구문

```
try
{
    정상적인 처리 내용
    예외 발생 경우 throw 전달인수;
}
catch(throw에서 전달받은 인수)
{
     예외 발생 수행할 내용
}
```

0으로 나누었을때 예외처리

```
try {
    if (n2 == 0)
        throw n1;
    quotient = n1 / n2;
    reminder = n1 % n2;
    cout << "몫 : " << quotient << endl;
    cout << "나머지 : " << reminder << endl;
}
catch (int e_n)
{
    cout << e_n << "은 0으로 나눌 수 없습니다!!!" << endl;
}
```

함수에서 예외 블록 호출하기

문자열을 정수로 변환하기

```
int stringToInt(const char x[]) {
    int sum = 0;
    int len = strlen(x);
    for (int i = 0; i < len; i++) {
        if (x[i] >= '0' && x[i] <= '9')
            sum = sum * 10 + x[i] - '0';
        else
            throw x;
    }
    return sum;
}
```

```
int n;
try {
    n = stringToInt("123");
    cout << "\"123\" 은 정수 " << n << "로 변환됨"
    n = stringToInt("1A3");
    cout << "\"1A3\" 은 정수 " << n << "로 변환됨"
}
catch (const char* s) {
    cout << s << " 처리에서 예외 발생!" << endl;
}
```