Inheritance of class

Recycle of class

- Class can be recycled by extending its definition
  - by adding member variables/functions to an existing class
  - Extension is regarded as inheritance
- **<u>Base</u>** class is the original class to be extended
  - also called as **super** class or **parent** class
- **<u>Derived</u>** class is the extended class from a base class
  - also called as **sub-**class or **child** class
  - Derived class refers the members declared in base class, where permission is controlled with access modifiers
  - Access modifier of a base class is mostly declared as public
- Multiple classes can be declared as base classes
  - Such multiple inheritance is prohibited in Java

Declare of derived class

```
class NameOfDerivedClass : AccessModifier　NameOfBaseClass　{
      Declaration of member variables/functions added in this class
};
```

実装例

```
class Point3D : public Point { // Extend 2D point into 3D point
public:
      int z; // variable of 3-rd dimension （x,y is defined in Point）
      Point3D() : Point() { z = 0; } // constructor with no parameter
      Point3D(int _x, int _y, int _z) : Point (_x, _y); // with param
      Point3D* median(Point3D p); // calculation of mid-point
};
```

Constructor of derived class

```
NameOfDerivedClass（Parameters） : ConstructorOfBaseClass
 （Parameters） {
     Process to be extended
}
```

- This format is only for a constructor
- Constructor of a base class is firstly called
- Call of a base class constructor can be omitted

Example：

```
Point3D (int _x, int _y, int _z) : Point (_x, _y) {
     z = _z; //x = _x; y = _y; is done in Point(_x, _y)
}
```

Access modifier for member variables

In a base class

```
class Point {
protected: // make public only for derived classes
    int x, y;
public:  // make public for all classes
    Point () { x = y = 0; } ···
```

In a derived class

```
class Point3D : public Point {
    int z; // x,y,z are used as member variables
public:  // make members below for all classes
    Point3D () : Point () { z = 0; } ···
```

Friend class

- Friend class is accessible independent of access modifiers（public, private, protected）

```
class Point {
friend class Triangle; // accessible from Triangle class
protected: // make public only for derived classes
    int x, y;
public:
    Point () { x = y = 0; } …
```

Usage in Triangle.h (or Triangle.cpp)

```
class Triangle {
public:
    Point pnt[3]; // data for 3 vertices
    Triangle () { // assign values to variables in Point
        pnt[0].x = pnt[0].y = 1;
        pnt[1].x = 1; pnt[1].y = 0;
    …}
```

Destructor

- Class can have its destructor
- Declare as ~NameOfClass() { process in deleting }

```
~Point() { std::cout << "(x,y)=" << x <<","<< y <<" deleted";}
~Point3D() { std::cout << "z=" << z << " deleted";}
```

- Explicitly called as, delete pointerOfObject;
- Destructor is also inherited by derived class
  - delete derivedClass ; firstly calls a destructor of derivedClass and then calls the destructor of its base

```
int main () {
    Point3D pointA (1,2,3), pointB(4,6,8);
    Point3D *med = pointA.median(pointB);
    delete med;
}
```

```
z=5 deleted (x,y)=2,4 deleted     for med
z=3 deleted (x,y)=1,2 deleted     for pointA  } Automatically called in
z=8 deleted  (x,y)=4,6 deleted    for  pointB } terminating a main function
```