Lecture 10

Object-Oriented Programming VI

Public and private labels, and friend declaration

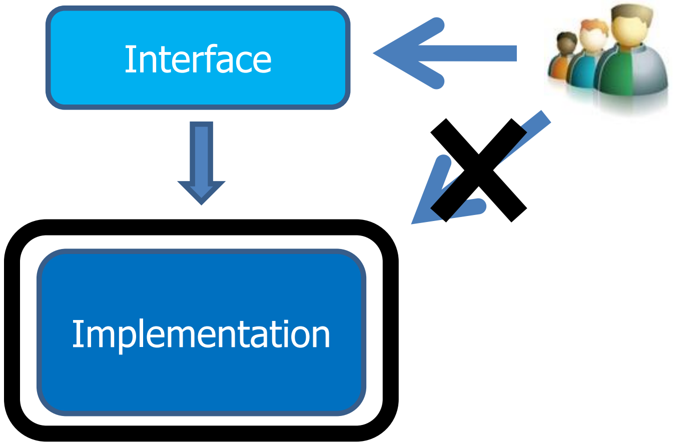
Prof. Hyeong-Seok Ko
Seoul National University
Graphics & Media Lab

Contents

- Encapsulation (12.1.2)
- Friend (12.5)

Encapsulation

- Encapsulation in C++
  - Hiding the implementation details of the data (and functions).
  - Data (and function) access is encouraged to be done through the member functions (interface).



- One important feature of OOP is encapsulation. Try to practice encapsulation.
  - The users of a class need to know only its **interface**.
    - The users need to know **what the class does** rather than **how the class is implemented**.

Public and Private Labels

- Access labels allows us to control the accessibility of the datafields and functions.
  - A public label notifies the start of a public section.
    - Members are accessible to all parts of the program.
  - A private label notifies the start of a private section.
    - Members are accessible only from the member functions of the class.
- Access labels can come multiple times.
- Access labels can be used to enforce encapsulation.

Enforcing Encapsulation

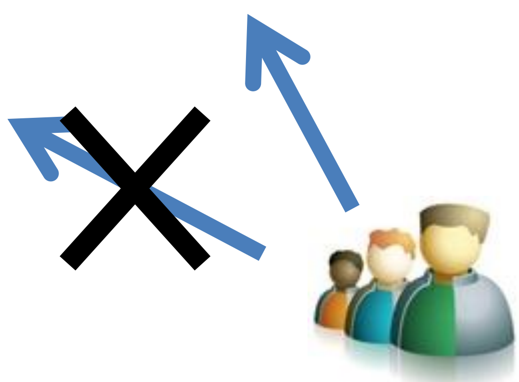
- Example (Image class)

```
class Image {
public:
   Image();
   Image(int width, int height);
   ~Image();

   void save_BMP(const char* filename) const;
   void read_BMP(const char* filename);

   void save_JPEG(const char* filename) const;
   void read_JPEG(const char* filename);

   void setColor(int i, int j, unsigned char red, unsigned char green, unsigned char blue);

private:
   unsigned char*      data;
   int                 width;
   int                 height;
};
```

Enforcing Encapsulation

- Example (Image class)

```
class Image {
public:
   Image();
   Image(int width, int height);
   ~Image();

   void save_BMP(const char* filename) const;
   void read_BMP(const char* filename);

   void save_JPEG(const char* filename) const;
   void read_JPEG(const char* filename);

   void setColor(int i, int j, unsigned char red, unsigned char green, unsigned char blue);
private:
   unsigned char*      data;
   int                 width;
   int                 height;
};
```

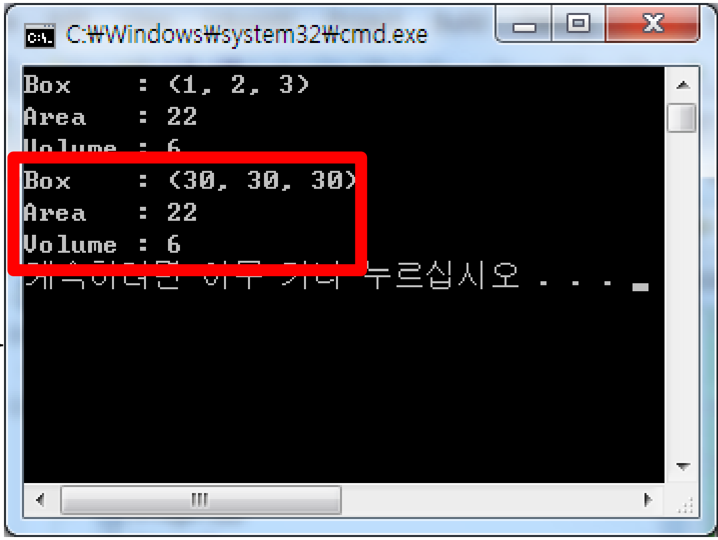


interfaces

actual datafields (hided)

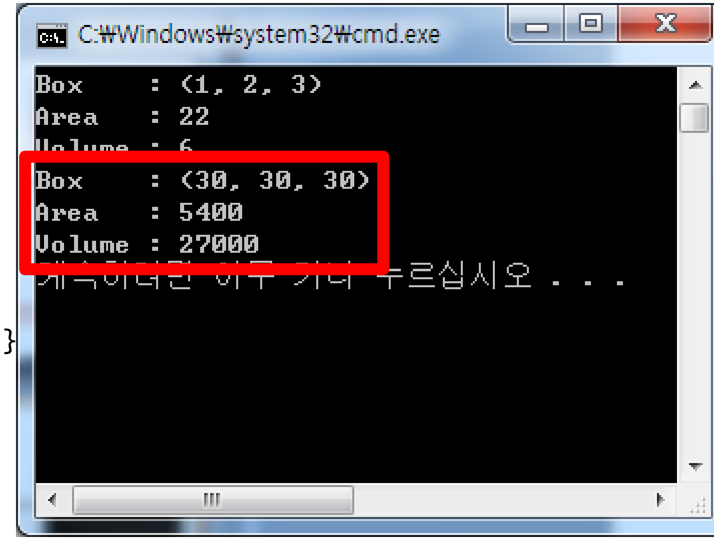
Why Encapsulation?

- Example (Box class)




```cpp
class Box {
public:
    Box() : height(0), width(0), length(0), area(0), volume(0){}

    void set(double h, double w, double l) {
        height = h; width = w; length = l;
        area = 2 * (h*w + w*l + l*h);
        volume = h*w*l;
    }
    void print() const {
        cout << "Box    : (" << height << ", " << width << ", " << length << ")" << endl;
        cout << "Area   : " << area << endl;
        cout << "Volume : " << volume << endl;
    }

    double height, width, length;
    double area, volume;
};

void main() {
    Box box1;
    box1.set(1,2,3);
    box1.print();
    box1.height = box1.width = box1.length = 30;  // ruins data consistency
    box1.print();
}
```

Why Encapsulation?

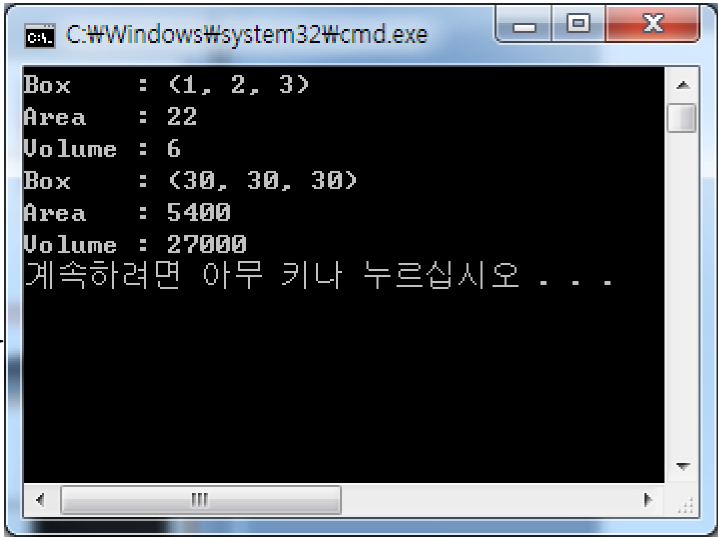


- Example (Box class)

```
class Box {
public:
   Box() : height(0), width(0), length(0), area(0), volume(0){}

   void set(double h, double w, double l) {
      height = h; width = w; length = l;
      area = 2 * (h*w + w*l + l*h);
      volume = h*w*l;
   }
   void print() const {
      cout << "Box    : (" << height << ", " << width << ", " << length << ")" << endl;
      cout << "Area   : " << area << endl;
      cout << "Volume : " << volume << endl;
   }

   double height, width, length;
   double area, volume;
};

void main() {
   Box box1;
   box1.set(1,2,3);
   box1.print();
   box1.set(30,30,30);   // data are now consistent
   box1.print();
}
```

Enforcing Encapsulation

- Example (Box class)




```
class Box {
public:
    Box() : height(0), width(0), length(0), area(0), volume(0){}

    void set(double h, double w, double l) {
        height = h; width = w; length = l;
        area = 2 * (h*w + w*l + l*h);
        volume = h*w*l;
    }
    void print() const {
        cout << "Box    : (" << height << ", " << width << ", " << length << ")" << endl;
        cout << "Area   : " << area << endl;
        cout << "Volume : " << volume << endl;
    }
private:
    double height, width, length;
    double area, volume;
};

void main() {
    Box box1;
    box1.set(1,2,3);
    box1.print();
    box1.set(30,30,30);    // box1.height=box1.width=box1.length=30; produces an error
    box1.print();
}
```

cannot access directly in main() function

Friend

- `friend` declaration allows a class to grant access to its nonpublic members to specified functions or classes.

```
#include <iostream>

class X {
public :
   X() {}


private :
   int a,b,c;
};

class Y {
public :
   Y() {}
   void func(X& var) {
      var.a = var.b = var.c = 0;  // Compilation Error !
                                  // cannot access private member declared in class 'X'
   }
};

int f(void) {
   X x;
   return x.a + x.b + x.c;        // Compilation Error !
}
```

Friend

- `friend` declaration allows a class to grant access to its nonpublic members to specified functions or classes.

```
#include <iostream>

class X {
public :
   X() {}
   friend class Y;
   friend int f(void);
private :
   int a,b,c;
};

class Y {
public :
   Y() {}
   void func(X& var) {
      var.a = var.b = var.c = 0;  // It is OK.
                                  // Class Y is a friend class of class X
   }
};

int f(void) {
   X x;
   return x.a + x.b + x.c;       // It is OK. Function f is a friend func of X.
}
```