# Topic 7
# Classes – Information Hiding

**private and public members**
**friend function and friend class**

# Information hiding

- Information Hiding refers to the characteristics of a module where inessential information of the module is hidden from the outside.
- Information Hiding can promote
  - Maintainability of class: Changes to the hidden members do not cause any additional modification of other classes
  - Understandability of classes: Only the exposed members need to be understood to

# Information Hiding in C++

```
class class-name {
  private:
    data members
  public:
    member functions
} ;
```

These private members cannot be accessed from outside of the class

# Information Hiding: An Example

```
# include <iostream>
using namespace std ;
class Rectangle {
private:
  int leftTopX, leftTopY ;
  int rightBottomX, rightBottomY ;
  void setLeftTop(int x, int y) {
    leftTopX = x ; leftTopY = y ;
  }
  void setRightBottom(int x, int y) {
     rightBottomX = x ; rightBottomY = y ;
  }
public:
  void set(int x1, int y1, int x2, int y2) {
    setLeftTop(x1, y1) ; setRightBottom(x2, y2) ;
  }
  void getLeftTop(int& x, int& y) {
    x = leftTopX ; y = leftTopY ;
  }
  void getRightBottom(int& x, int& y) {
    x = rightBottomX ; y = rightBottomY ;
  }
  int getArea() {
    return (rightBottomX - leftTopX) *
      (rightBottomY - leftTopX) ;
  }
} ;
```

```
int main() {
  int x1, y1, x2, y2 ;
  cin >> x1 >> y1 >> x2 >> y2 ;

  Rectangle r1 ;
  r1.set(x1, y1, x2, y2) ;          // OK
  r1.leftTopX = r1.leftTopX + 1 ; // ERROR

  int x3, y3, x4, y4 ;
  r1.getLeftTop(x3, y3) ;
  r1.getRightBottom(x4, y4) ;

  Rectangle r2 ;
  r2.setLeftTop(x3, y3) ;          // ERROR
  r2.setRightBottom(x4, y4) ; // ERROR
  r2.set(x3, y3, x4, y4) ;          // OK

  cout << endl << r1.getArea() << '\t'
    << r2.getArea() << endl ;
}
```

# 클래스의 정의: Rectangle.h

```
# ifndef __RECTANGLE_H
# define __RECTANGLE_H

class Rectangle {
private: // private is default
    int leftTopX, leftTopY ;
    int rightBottomX, rightBottomY ;
public:
    // now setLeftTop(), setRightBottom() can be invoked from outside
    void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
    void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

    void set(int x1, int y1, int x2, int y2) { setLeftTop(x1, y1) ; setRightBottom(x2, y2) ; }
    void getLeftTop(int& x, int& y) const { x = leftTopX ; y = leftTopY ; }
    void getRightBottom(int& x, int& y) const { x = rightBottomX ; y = rightBottomY ; }

    int getWidth() const { return rightBottomX - leftTopX ; }
    int getHeight() const { return rightBottomY - leftTopY ; }

    int getArea() const ;
    void moveBy(int deltaX, int deltaY) ;
} ;
# endif
```

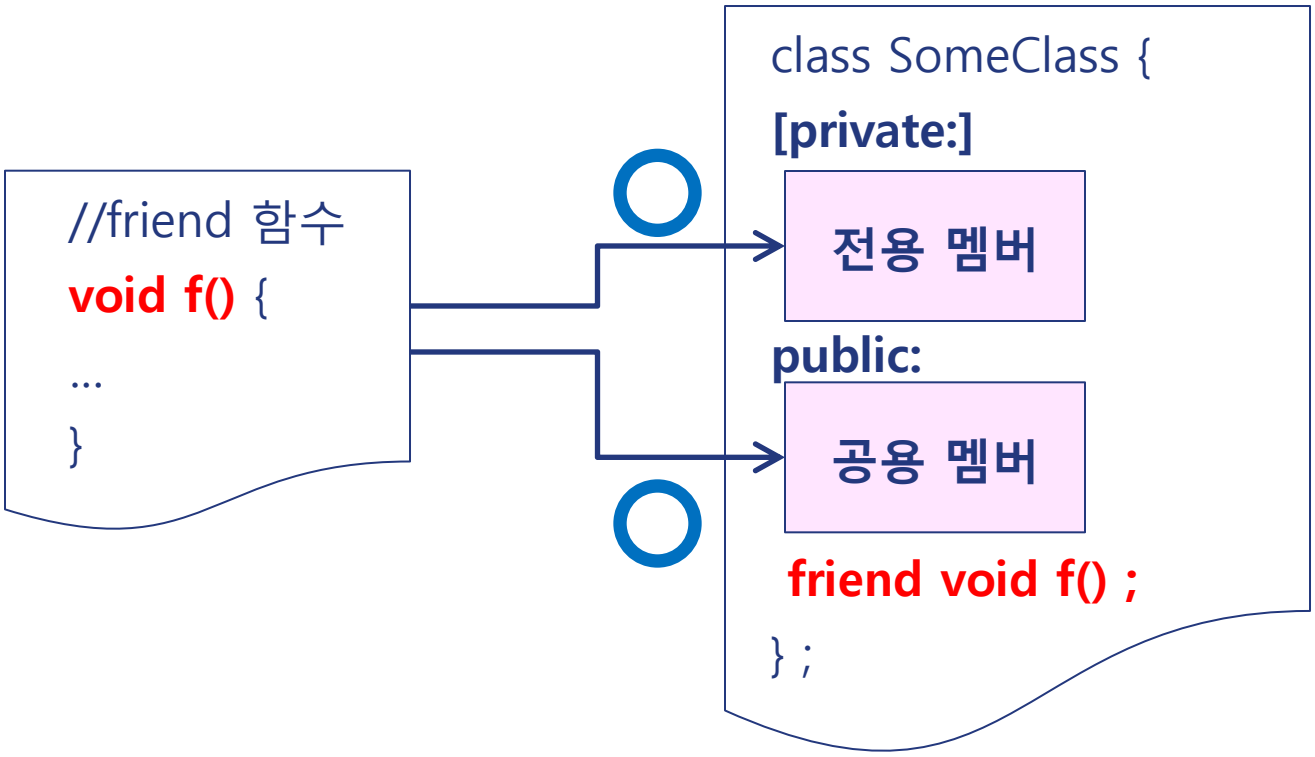
# Rectangle.cpp

```
// Rectangle.cpp

# include "Rectangle.h"

int Rectangle::getArea() const {
    return getWidth() * getHeight() ;
}

void Rectangle::moveBy(int deltaX, int deltaY) {
    setLeftTop(leftTopX+deltaX, leftTopY+deltaY) ;
    setRightBottom(rightBottomX+deltaX, rightBottomY+deltaY) ;
}
```

# friend

- friend allows classes outside to access private members.
- Therefore, friend violates information hiding principle.
- friend creates tight coupling between the classes and should be used sparingly
- Three forms of friend are allowed:
  1. friend non-member function
  2. friend class
  3. friend member function

# Friend function



# friend non-member function

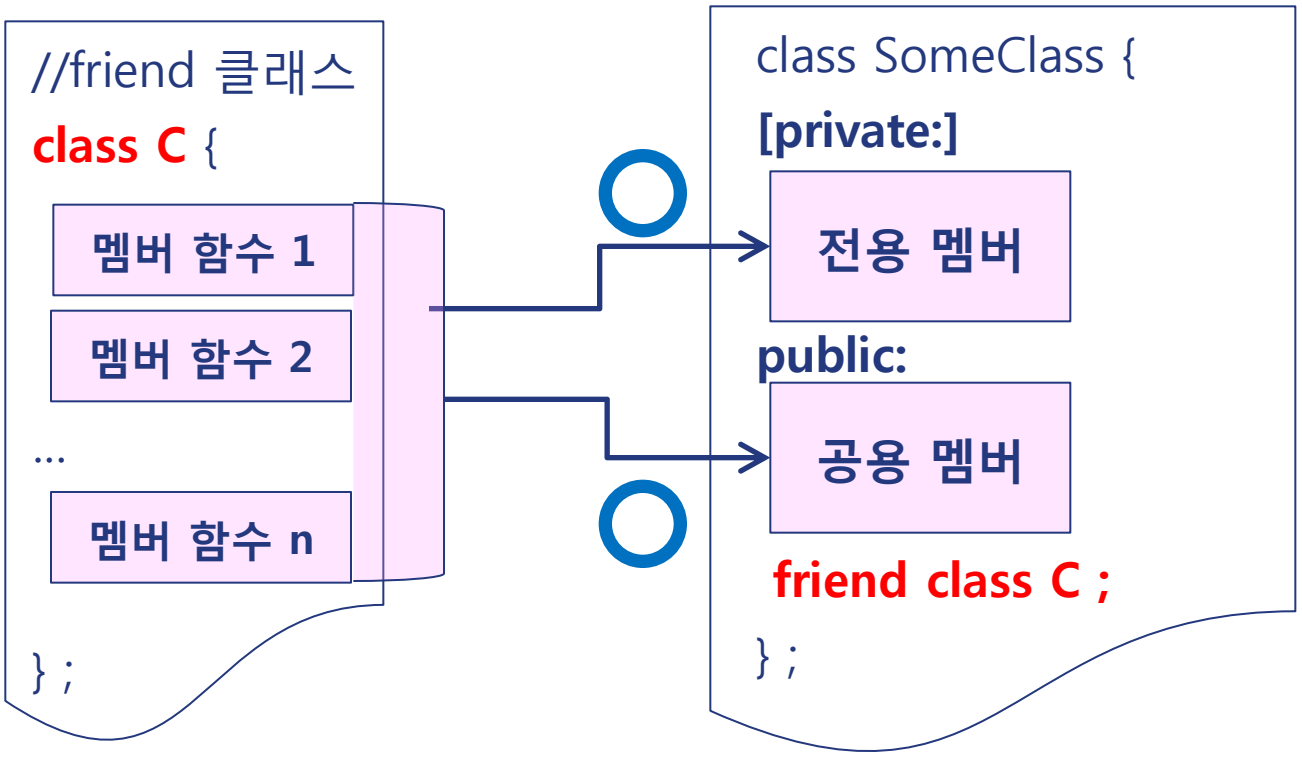
- A non-member function can be declared as a friend.

```
class Window {
  private:
    // ...
    string title ;
    // ...
  public:
    string getTitle() const ;
    friend void friendOfWindow() ;
} ;
```

```
void friendOfWindow(
      const Window& anWindow) {
   cout << anWindow.title ;
}

void nonFriendOfWindow(
      const Window& anWindow) {
   cout << anWindow.title ; // ERROR
   cout << anWindow.getTitle() ;
}
```

friendOfWindow() can access the private member of Window

# Friend class



# friend class

- Every members of a friend class can access private members.

```
class StringNode{
  string data;
  StringNode* next ;

  StringNode(const string& d="") : data(d) { next = 0 ;}
  bool isEqual(const StringNode& n) const {
    return data == n.data ;
  }

  friend class StringList ;
} ;
```

```
class StringList {
    StringNode* head ; // default is private
  public:
    StringList() { head = '₩0' ; }
    void addNode(const StringNode& node) {
      StringNode* newNode = new StringNode(node) ;
      if ( head == '₩0' ) head = newNode ;
      else {
        head->next = newNode ;
        head = newNode ;
      }
    }
    void removeNode(const StringNode& node) {
      StringNode* cur = head, * prev = '₩0' ;
      while ( cur != '₩0' ) {
        if ( next->isEqual(node) ) {
          if ( prev ) prev->next = cur->next ;
          else head = cur->next ;
          delete cur ;
          break ;
        }
        cur = cur->next ;
      }
  }
} ;
```

# friend member function

- A particular member function can be a friend

```
class StringNode{
  private:
    string data;
    StringNode* next ;
  public:
    bool isEqual() const ;
    StringNode* getNext() const ;
    void setNext
      (const StringNode* const) ;
   friend void addNode
    (const StringNode& node) ;
} ;
```

```
class StringList {
    StringNode* head ; // default is private
  public:
    void addNode(const StringNode& node) {
      StringNode* newNode = new StringNode ;
      newNode->data = node.data ;
      newNode->next = '₩0' ;
      if ( head == '₩0' ) head = newNode ;
      else {
        head->next = newNode ;
        head = newNode ;
      }
    }
    void removeNode(const StringNode& node) {
      StringNode* cur = head, * prev = '₩0' ;
      while ( cur != '₩0' ) {
        if ( next->isEqual(node) ) {
          if ( prev ) prev->setNext(cur->getNext()) ;
          else head = cur->getNext() ;
          delete cur ;
          break ;
        }
        cur = cur->getNext() ;
      }
}
```