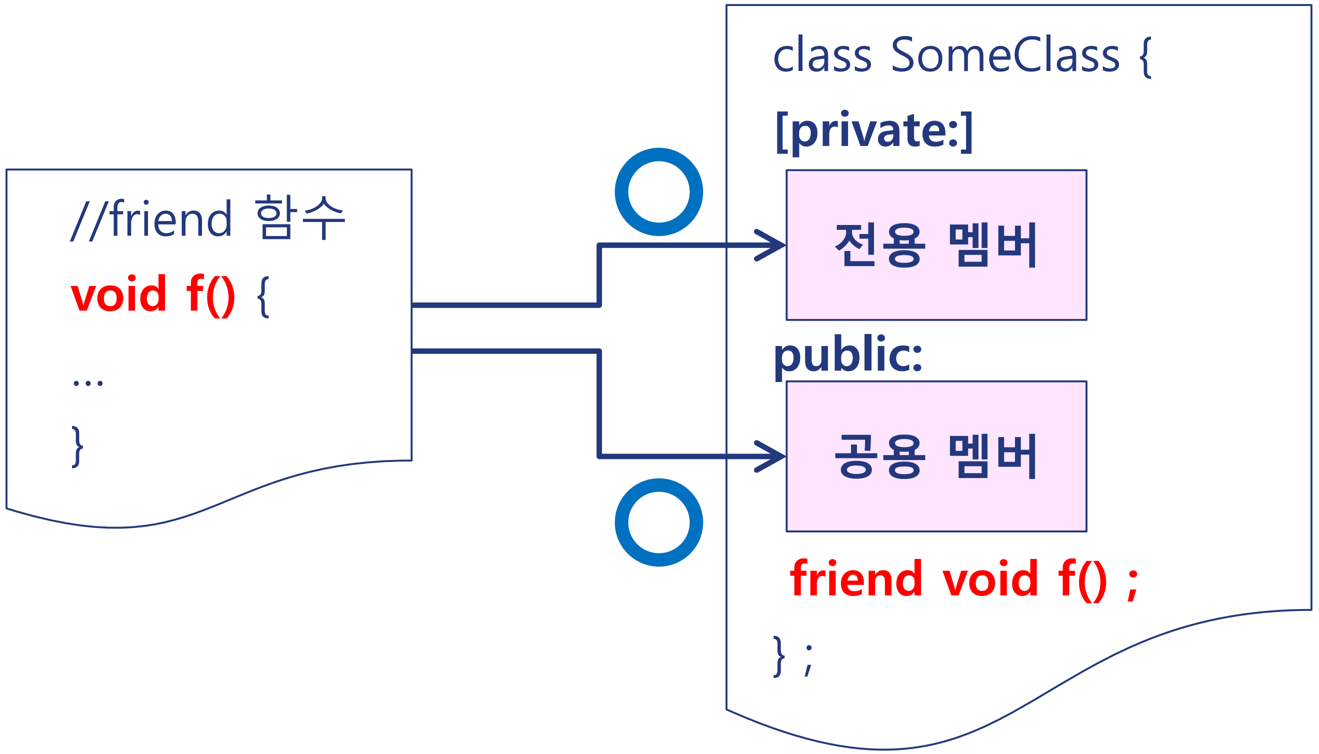
friend

- 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 - Member Function

- A particular member function can be a friend

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

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

friend - Non-member Function

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

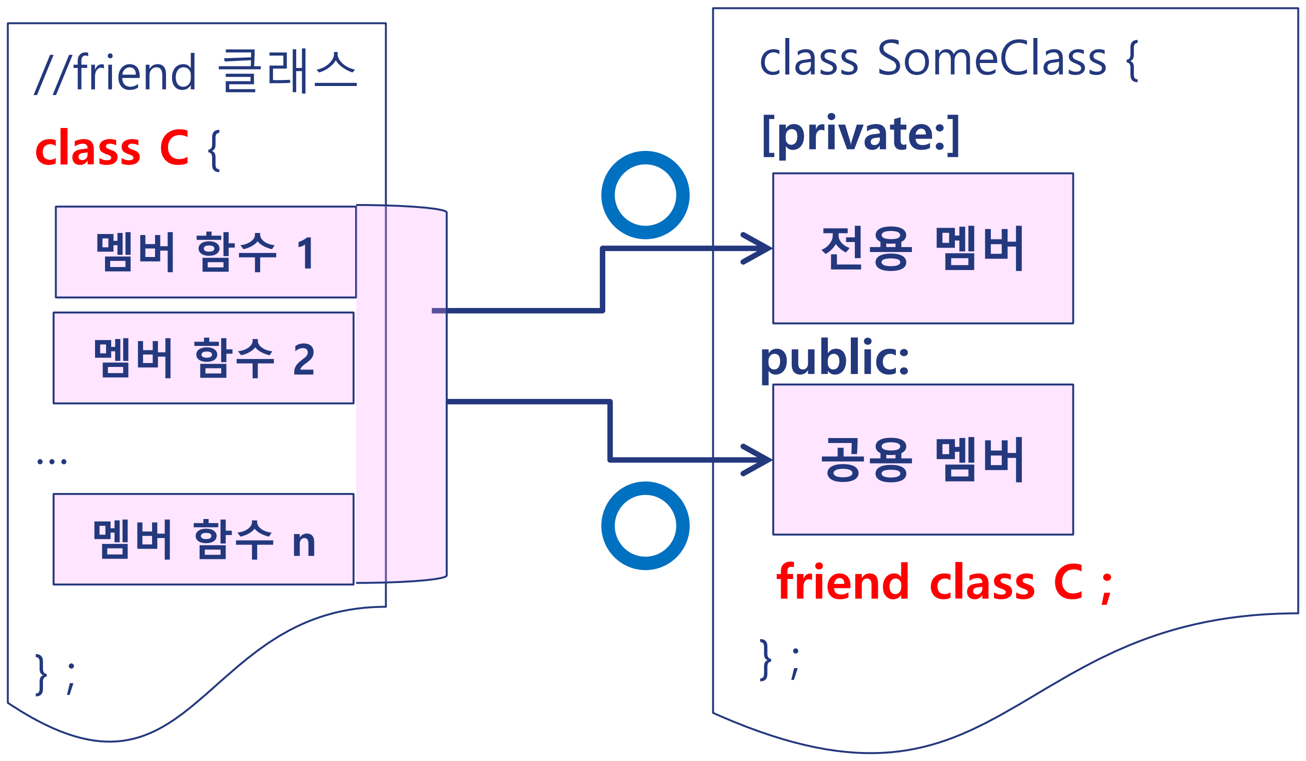
```
class Window {
  private:
    // ...
    string title ;
    // ...
  public:
     string getTitle() const ;
     friend void friendOfWindow(const Window&) ;
} ;
```

friendOfWindow() can access the private member of Window

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

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

friend Class



friend Class

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

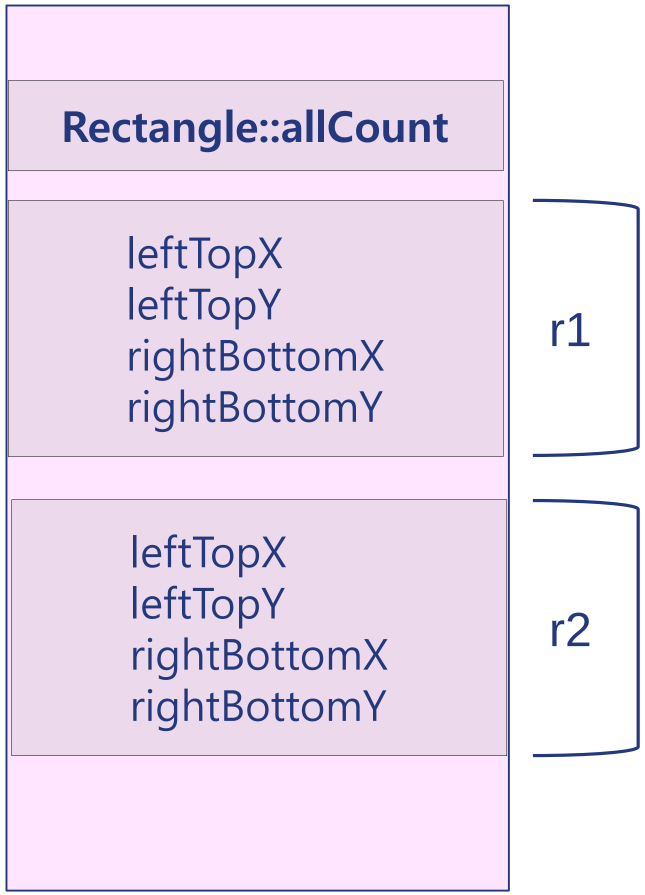
```
class StringNode {
  string data;
  StringNode* next {nullptr};

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

  friend class StringList ;
} ;
```

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

static Data Member



static Data Member

```
// Rectangle.h
# ifndef __RECTANGLE_H
# define __RECTANGLE_H

class Rectangle {
   int leftTopX, leftTopY ;
   int rightBottomX, rightBottomY ;
public:
   static int allCount ;

   void setLeftTop(int x, int y) {
      leftTopX = x ; leftTopY = y ;
   }
   void setRightBottom(int x, int y) {
      rightBottomX = x ; rightBottomY = y ;
   }
   ...
} ;
# endif
```

```
// Rectangle.cpp
# include "Rectangle.h"

int Rectangle::allCount = 0 ;
...
```

```
# include <iostream>
# include "Rectangle.h"
using namespace std ;

int main() {
   Rectangle r1 ;
   r1.set(1, 1, 2, 2) ;
   cout << Rectangle::allCount << endl ;

   Rectangle r2 ;
   r2.set(10, 10, 20, 20) ;
   cout << Rectangle::allCount << endl ;
}
```

```
# ifndef __RECTANGLE_H
# define __RECTANGLE_H

class Rectangle {
   int leftTopX, leftTopY ;
   int rightBottomX, rightBottomY ;
public:
   static int allCount ;

   // 생성자: 객체 생성시 자동 호출됨
   Rectangle() { allCount ++ ; }
   // 소멸자: 객체 소멸시 자동 호출됨
   ~Rectangle() { allCount -- ; }
   …
} ;
# endif
```

```
# include "Rectangle.h"


int Rectangle::allCount = 0 ;


…
```

```
# include <iostream>
# include "Rectangle.h"
using namespace std ;

Rectangle gRectangle1, gRectangle2 ;

int main() {
     cout << Rectangle::allCount << endl ;
     Rectangle r1 ;
     cout << Rectangle::allCount << endl ;

     for ( int i = 0 ; i < 3 ; i ++ ) {
          Rectangle r ;
          cout << Rectangle::allCount << endl ;
     }

     Rectangle* pR = new Rectangle ;
     cout << Rectangle::allCount << endl ;
     delete pR ;
     cout << Rectangle::allCount << endl ;
}
```

static Member Function

```
# ifndef __RECTANGLE_H
# define __RECTANGLE_H

class Rectangle {
   static int allCount ;  //default is private
   int leftTopX, leftTopY ;
   int rightBottomX, rightBottomY ;
public:
   // 정적 데이터멤버만 호출가능함
   static int getAllCount() { return allCount ; }
   static bool noRectangle() { return allCount == 0 ; }
   Rectangle() { allCount ++ ; }
   ~Rectangle() { allCount -- ; }
   …
} ;
# endif
# include <iostream>
# include <vector>
# include <string>
# include "Rectangle.h"
using namespace std ;
int main() {
  vector<Rectangle*> rectangles ;
  do {
    string command ;
    cin >> command ;
    if ( command == "ADD" )
      rectangles.push_back(new Rectangle) ;
    else if ( command == "DELETE" ) {
      vector<Rectangle*>::iterator head = rectangles.begin() ;
      Rectangle* r = *head ;
      delete r ;
      rectangles.erase(head) ;
    }
    else break ;
    cout << Rectangle::getAllCount() << endl ;
  } while ( Rectangle::noRectangle() == false ) ;
  for (vector<Rectangle*>::iterator Iter = rectangles.begin( ); Iter != rectangles.end( ); Iter++) {
    Rectangle* r = *Iter ;
    delete r ;
  }
}
```

Good Design: 싱글톤 (Singleton)

- 단 하나의 인스턴스만을 만들어야 하는 경우
- 싱글톤 방식과 모노스테이트(모든 멤버 변수가 static) 방식이 있음

```
class Student {
public:
        static Student& getInstance() {
                static Student s;              // static 지역변수를 반환함!
                return s;
        }
        Student (Student const&) = delete;
        Student (Student&&) = delete;
        Student & operator=(Student const&) = delete;
        Student & operator=(Student &&) = delete;
}
```

- 장점만큼 단점도 많아 싱글톤을 사용할 때는 사용목적에 맞도록 세심한 주의가 필요함

const Member Function

```
# ifndef __RECTANGLE_H
# define __RECTANGLE_H
class Rectangle {
public:
   static int allCount ;
   int leftTopX, leftTopY ;
   int rightBottomX, rightBottomY ;

   Rectangle() { allCount ++ ; }
   ~Rectangle() { allCount -- ; }
   static int getAllCount() { return allCount ; } // not const
   static bool noRectangle() { return allCount == 0 ; }
   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
# include "Rectangle.h"
int Rectangle::allCount = 0 ;
int Rectangle::getArea() const { return getWidth() * getHeight() ; }
void Rectangle::moveBy(int deltaX, int deltaY) {
  setLeftTop(leftTopX+deltaX, leftTopY+deltaY) ;
  setRightBottom(rightBottomX+deltaX, rightBottomY+deltaY) ;
}
```

```
# include <iostream>

# include "Rectangle.h"
using namespace std ;

void readRectangle(Rectangle& r) {
   int x1, y1, x2, y2 ;
   cin >> x1 >> y1 >> x2 >> y2 ;
   r.setLeftTop(x1, y1) ; r.setRightBottom(x2, y2) ;
}
void printRectangle(const Rectangle& r) {
   int x1, y1, x2, y2 ;
   r.getLeftTop(x1, y1) ; r.getRightBottom(x2, y2) ;
   cout << x1 << '₩t' << y1 << '₩t' << x2 << '₩t' << y2 << endl ;
   // r.setLeftTop(0, 0) ; // ERROR
}
int main() {
   Rectangle r ;
   readRectangle(r) ;
   printRectangle(r) ;
}
```