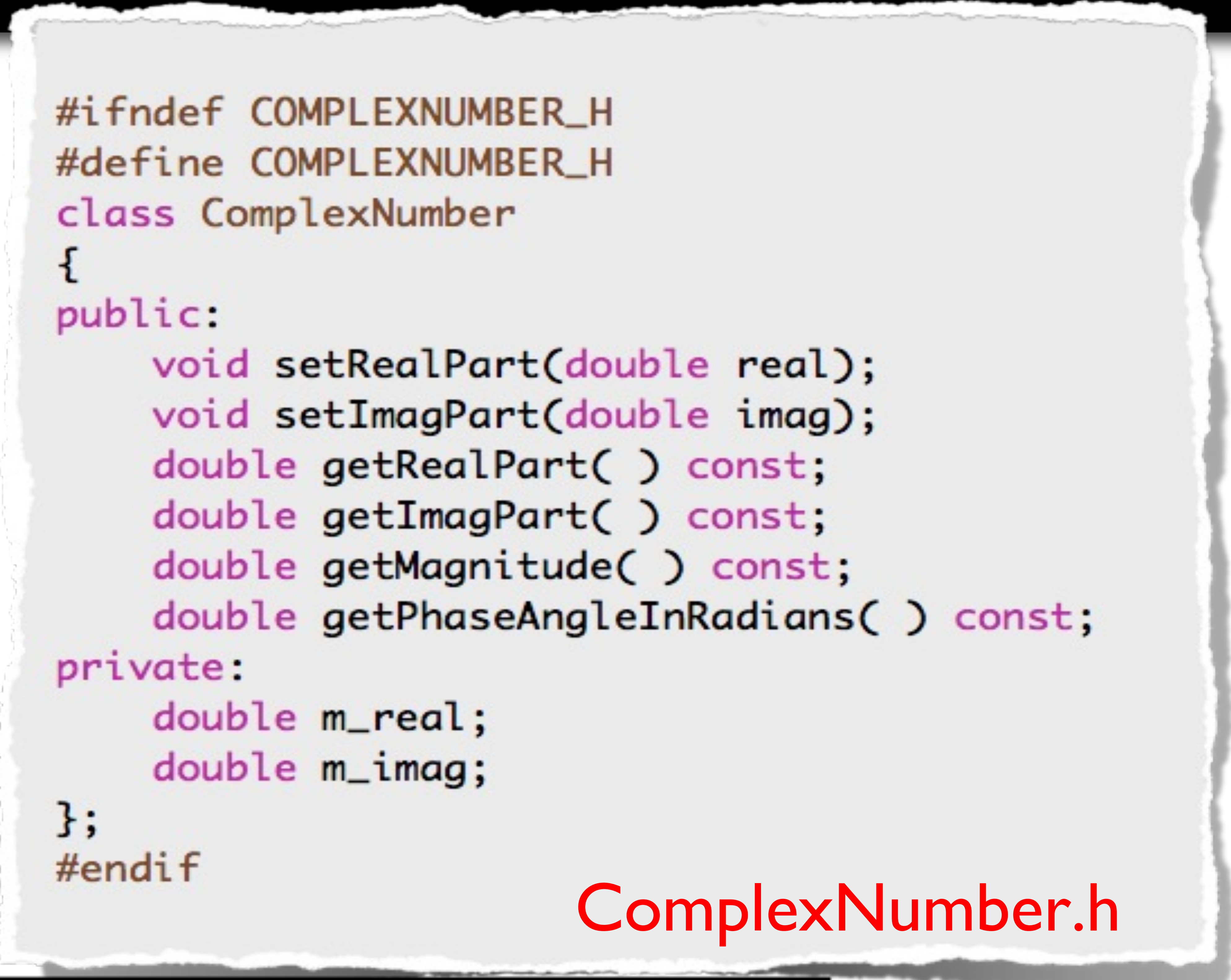
Lecture: Classes Using Multiple Files

ENGR 2730:Computers in Engineering

Defining a class in C++ Using Multiple Files

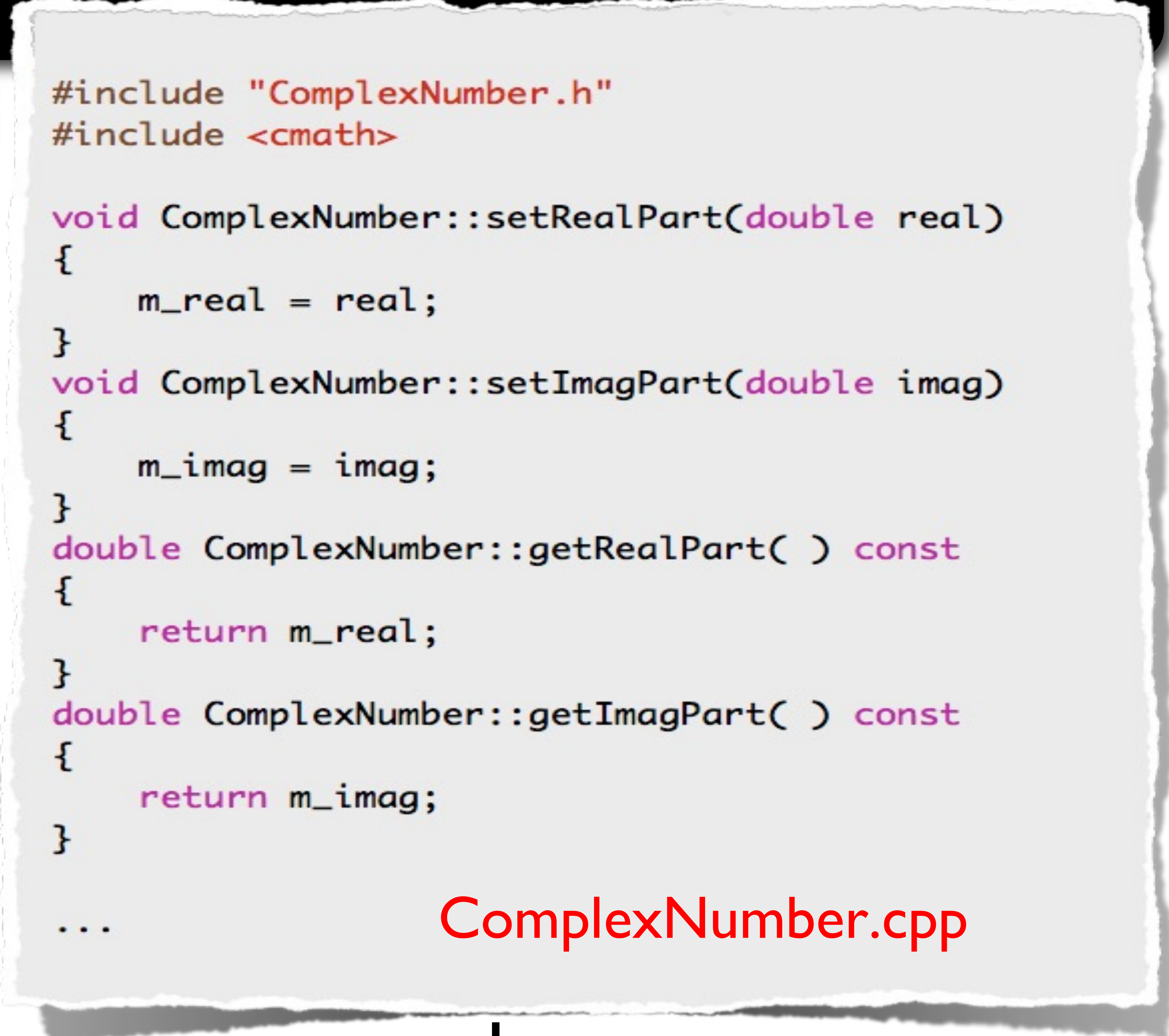
Example: A **partial** ComplexNumber class

class
“interface”



```cpp
#ifndef COMPLEXNUMBER_H
#define COMPLEXNUMBER_H
class ComplexNumber
{
public:
    void setRealPart(double real);
    void setImagPart(double imag);
    double getRealPart( ) const;
    double getImagPart( ) const;
    double getMagnitude( ) const;
    double getPhaseAngleInRadians( ) const;
private:
    double m_real;
    double m_imag;
};
#endif
```

ComplexNumber.h



```cpp
#include "ComplexNumber.h"
#include <cmath>

void ComplexNumber::setRealPart(double real)
{
    m_real = real;
}
void ComplexNumber::setImagPart(double imag)
{
    m_imag = imag;
}
double ComplexNumber::getRealPart( ) const
{
    return m_real;
}
double ComplexNumber::getImagPart( ) const
{
    return m_imag;
}

...
```

ComplexNumber.cpp

class
“implementation”

main.cpp

```cpp
#include <iostream>
#include "ComplexNumber.h"
int main()
{
    ComplexNumber c1;
    c1.setRealPart(3);
    c1.setImagPart(4);
    std::cout << c1.getMagnitude() << std::endl;
}
```

instantiated object

Example: A **partial** ComplexNumber class

Class "Interface"

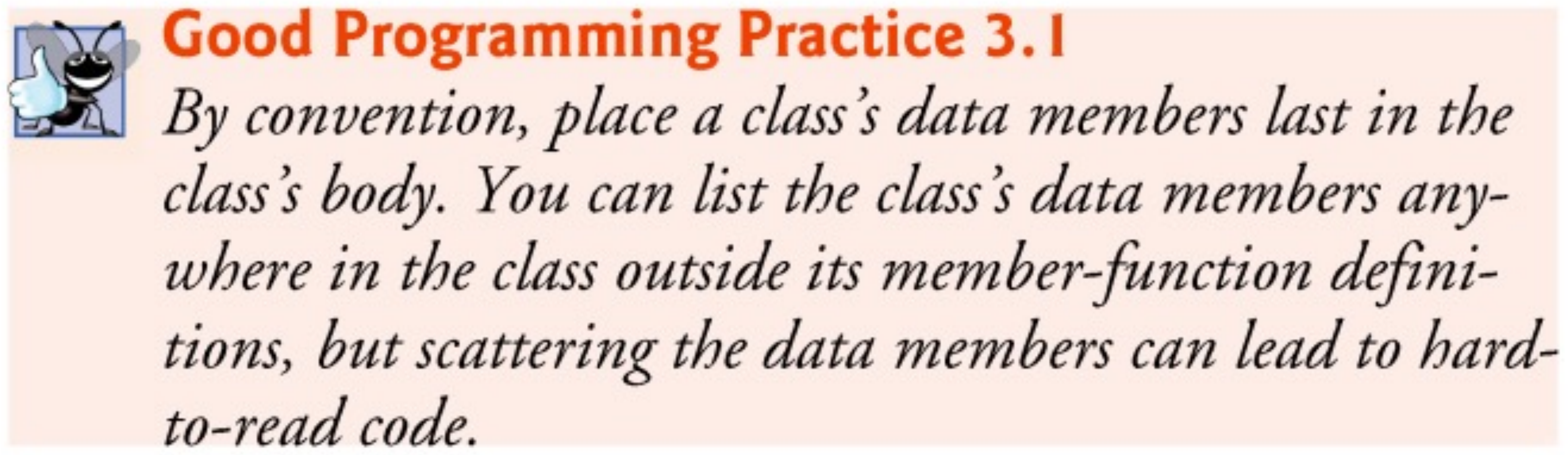
```
class ComplexNumber
{
public:
    void setRealPart(double real); // sets the real part of the complex number
    double getRealPart( ) const;   // returns the real part of the complex number

    void setImagPart(double imag);
    double getImagPart( ) const;  // returns the imaginary part of the complex number

    double getMagnitude( ) const; // returns the magnitude of the complex number

    double getPhaseAngleInRadians( ) const; // returns the phase angle of the complex number (in radians)

private:
    double m_real; // real part of complex number
    double m_imag; // imaginary part of complex number
};
```



Good Programming Practice 3.1

By convention, place a class's data members last in the class's body. You can list the class's data members anywhere in the class outside its member-function definitions, but scattering the data members can lead to hard-to-read code.

Example: A **partial** ComplexNumber class

Class “Interface”

member functions

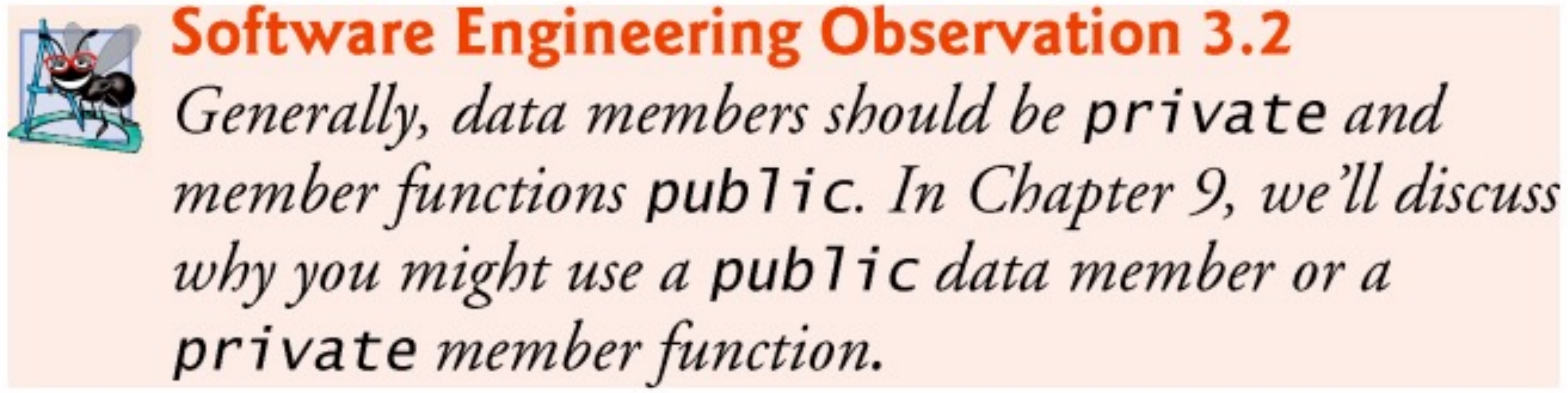
data members

```
class ComplexNumber
{
public:
    void setRealPart(double real); // sets the real part of the complex number
    double getRealPart( ) const;   // returns the real part of the complex number

    void setImagPart(double imag);
    double getImagPart( ) const;  // returns the imaginary part of the complex number


    double getMagnitude( ) const; // returns the magnitude of the complex number

    double getPhaseAngleInRadians( ) const; // returns the phase angle of the complex number (in radians)
private:
    double m_real; // real part of complex number
    double m_imag; // imaginary part of complex number
};
```



Software Engineering Observation 3.2

Generally, data members should be `private` *and member functions* `public`*. In Chapter 9, we'll discuss why you might use a* `public` *data member or a* `private` *member function.*

Separating the interface from the implementation

ComplexNumber.h

```
class ComplexNumber
{
public:
    void setRealPart(double real);
    double getRealPart( ) const;

    void setImagPart(double imag);
    double getImagPart( ) const;

    // returns the magnitude of the complex number
    double getMagnitude( ) const;

    // returns the phase angle of the complex number (in radians)
    double getPhaseAngleInRadians( ) const;

private:
    double m_real; // real part of complex number
    double m_imag; // imaginary part of complex number
};
```

binary scope resolution operator

::

ComplexNumber.cpp

```
void ComplexNumber::setRealPart(double real)
{
    m_real = real;
}

void ComplexNumber::setImagPart(double imag)
{
    m_imag = imag;
}

(etc.)
```

In-Class Assignment

- Translate the previous Circle class example into three files.

Adding New Header and Source Files to CLion Project

- Instructions for Adding Header and Source Files
  - Right Click on Project Folder in Left Navbar.
  - Select New->C/C++ Source File.
  - Check the Create an associated header box.
  - Enter a Name (e.g., Circle) for the files and click OK.
  - Add #include statement at the top of the main.cpp to include the header file.
  - Add header and source file to SVN.

- What Happens
  - A Circle.h and Circle.cpp get added to the project
  - Both files get added to the CMakeLists.txt file

Circle Class Defined in Multiple Files

```
#ifndef HOMEWORK_1_CIRCLE_H
#define HOMEWORK_1_CIRCLE_H

class Circle{
public:
    Circle(double radius = 0);

    double getRadius() const;
    void setRadius(double radius);

private:
    double m_radius;
};

#endif
```

Circle.h

```
#include "Circle.h"

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

double Circle::getRadius() const {
    return m_radius;
}

void Circle::setRadius(double radius) {
    if (radius >= 0){
        m_radius = radius;
    } else {
        m_radius = 0;
    }
}
```

Circle.cpp

```
#include <iostream>
#include "Circle.h"

using namespace std;

int main() {
    Circle c1;
    Circle c2(5);
    Circle c3(-5);

    cout << c1.getRadius();
    cout << c2.getRadius();
    cout << c3.getRadius();

    return 0;
}
```

main.cpp

Output:

```
radius = 0
radius = 5
radius = 0
```

Circle Class Defined in Multiple Files

```
#ifndef HOMEWORK_1_CIRCLE_H
#define HOMEWORK_1_CIRCLE_H

class Circle{
public:
    Circle(double radius = 0);

    double getRadius() const;
    void setRadius(double radius);

private:
    double m_radius;
};

#endif
```

Constructors have no return type.

Principle of Least Privilege: const prevents getRadius() from changing the value of m_radius.

Default radius value if no argument is given to constructor.

Circle.h

Circle Class Defined in Multiple Files

Must include header file.

Constructors have no return type.

Use setter in constructor.

Must prefix each function with the name of the class, i.e., Circle::

Setter methods are responsible for attribute validity, i.e., radius must be positive.

```cpp
#include "Circle.h"

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

double Circle::getRadius() const {
    return m_radius;
}

void Circle::setRadius(double radius) {
    if (radius >= 0){
        m_radius = radius;
    } else {
        m_radius = 0;
    }
}
```

Circle.cpp

Circle Class Defined in Multiple Files

Must include header file.

Must use getter to get the value of the radius. c1.m_radius does not work since m_radius is defined as private.

```cpp
#include <iostream>
#include "Circle.h"

using namespace std;

int main() {
    Circle c1;
    Circle c2(5);
    Circle c3(-5);

    cout << c1.getRadius();
    cout << c2.getRadius();
    cout << c3.getRadius();

    return 0;
}
```

main.cpp

```
Output:
radius = 0
radius = 5
radius = 0
```

Commenting Classes

- All class methods/functions and attributes/data members need to be commented.
- At a minimum, the comments for a method must include descriptions of the input and output variables and a description of what the method does.
- For more complicated methods, you should put your name, date that you wrote the method and a brief description of the algorithm used in the method.
- For this class, you can write your documentation for each function in the .cpp or .h file. In practice, the documentation should be put in the header file.

Account Class

```
class Account {
public:
    // Account constructor with two parameters
    Account(string accountName, int initialBalance);

    // function that deposits (adds) only a valid amount to the balance
    void deposit(int depositAmount);

    int getBalance() const { return m_balance; }

    void setName(string accountName) {m_name = accountName;}
    string getName() const { return m_name;}

private:
    string m_name;     // name of account
    int m_balance = 0; // account balance

};
```

Constructor requires exactly two input parameters.

Initialize m_balance to zero.

Account.h

Account Class

```
#include "Account.h"

Account::Account(string accountName, int initialBalance)
    : m_name{accountName}  // Call the constructor for the string m_name
{
    // Validate that the initial balance is > 0. If not,
    // m_balance keeps it's default value of 0.
    if (initialBalance > 0){
        m_balance = initialBalance;
    }
}

void Account::deposit(int depositAmount) {
    if (depositAmount > 0){
        m_balance += depositAmount;
    }
}
```

Member Initializer list syntax

Member Initializer list calls the constructors of the class data members

Account.cpp

Account Class

```cpp
int main() {
    Account a1{"Jane Green", 50};
    Account a2{"John Blue", -7};

    cout << a1.getName() << ": balance = $"  << a1.getBalance() << endl;
    cout << a2.getName() << ": balance = $"  << a2.getBalance() << endl;

    a1.deposit(75);
    a2.deposit(150);

    cout << a1.getName() << ": balance = $"  << a1.getBalance() << endl;
    cout << a2.getName() << ": balance = $"  << a2.getBalance() << endl;

    return 0;
}
```

main.cpp

```
Output:
Jane Green: balance = $50
John Blue: balance = $0
Jane Green: balance = $125
John Blue: balance = $150
```