

02: Introduction to Classes and Objects

Programming Technique II
(SCSJ1023)

Adapted from Tony Gaddis and Barret Krupnow (2016), Starting out with C++: From Control Structures through Objects

Content

 Defining classes

 Creating Object

 Private Members

- Why have private members ?
- Using private members function

 Separating Class Specification from Implementation

 Inline Member Functions

Defining Classes

Defining classes

✿ Classes are defined using keyword class, with the following syntax:

```
class ClassName
{
    declaration;
    declaration;
};
```

✿ The declaration statements inside a class declaration are for the variables/attributes and functions/methods that are members of the class


Defining Class : Example

```
class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
        double getArea() const;
};
```

Defining Class : Access Specifiers

 Used to control access to members of the class

 public: can be accessed by functions outside of the class

 private: can only be called by or accessed by functions that are members of the class

Defining Class with Access Specifiers : Example

```
class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
        double getArea() const;
};
```

Private Members

Public Members

More on Access Specifiers

✿ Can be listed in any order in a class

✿ Can appear multiple times in a class

✿ If not specified, the default is private

Defining a Member Function




When **defining** a member function:

- ◆ Put **prototype** in class declaration
- ◆ Define **function/method** using class name and scope resolution operator (::)

Example


```
void Rectangle::setWidth(double w)
{
    width = w;
}
```

Using `const` With Member Functions

 `const` appearing after the parentheses in a member function declaration specifies that the function will not change any attribute in the calling object.

```
double getWidth() const;  
double getLength() const;  
double getArea() const;
```

Accessors and Mutators

 **Mutator:** a member function that stores a value in a private member variable (attribute), or **changes** its value in some way.

 **Accessor:** function that retrieves a value from a private member variable. Accessors **do not change** an object's attribute, so they should be marked const.

Creating Object

Creating Object

✿ An **object** is an **instance** of a class

✿ To **define** an object - defined like structure variables:

```
Rectangle r;
```

✿ **Access members** using dot operator:

```
r.setWidth(5.2);  
cout << r.getWidth();
```

Compiler error if attempt to access **private** member using dot operator

Example: Define Class and Object

Program 2-1

```
1 // This program demonstrates a simple class.
2 #include <iostream>
3 using namespace std;
4
5 // Rectangle class declaration.
6 class Rectangle
7 {
8     private:
9         double width;
10        double length;
11    public:
12        void setWidth(double);
13        void setLength(double);
14        double getWidth() const;
15        double getLength() const;
16        double getArea() const;
17 };
18
19 //*****
20 // setWidth assigns a value to the width member.  *
21 //*****
22
23 void Rectangle::setWidth(double w)
24 {
25     width = w;
26 }
27
28 //*****
29 // setLength assigns a value to the length member. *
30 //*****
31
```

Class declaration

Example: Define Class and Object

Program 2-1 (Continued)

```
32 void Rectangle::setLength(double len)
33 {
34     length = len;
35 }
36
37 //*****
38 // getWidth returns the value in the width member. *
39 //*****
40
41 double Rectangle::getWidth() const
42 {
43     return width;
44 }
45
46 //*****
47 // getLength returns the value in the length member. *
48 //*****
49
50 double Rectangle::getLength() const
51 {
52     return length;
53 }
54
```

Program 2-1 (Continued)

**objects
definition**

```
55  //*****
56  // getArea returns the product of width times length. *
57  //*****
58
59  double Rectangle::getArea() const
60  {
61      return width * length;
62  }
63
64  //*****
65  // Function main *
66  //*****
67
68  int main()
69  {
70      Rectangle box;    // Define an instance of the Rectangle class
71      double rectWidth; // Local variable for width
72      double rectLength; // Local variable for length
73
74      // Get the rectangle's width and length from the user.
75      cout << "This program will calculate the area of a\n";
76      cout << "rectangle. What is the width? ";
77      cin >> rectWidth;
78      cout << "What is the length? ";
79      cin >> rectLength;
80
81      // Store the width and length of the rectangle
82      // in the box object.
83      box.setWidth(rectWidth);
84      box.setLength(rectLength);
```


Program 2-1 (Continued)

```
85
86     // Display the rectangle's data.
87     cout << "Here is the rectangle's data:\n";
88     cout << "Width: " << box.getWidth() << endl;
89     cout << "Length: " << box.getLength() << endl;
90     cout << "Area: " << box.getArea() << endl;
91     return 0;
92 }
```

Program Output

This program will calculate the area of a rectangle. What is the width? **10 [Enter]**
What is the length? **5 [Enter]**
Here is the rectangle's data:
Width: 10
Length: 5
Area: 50

Private Members

Private Members

Why have private members ?

 Making data members **private** provides data protection

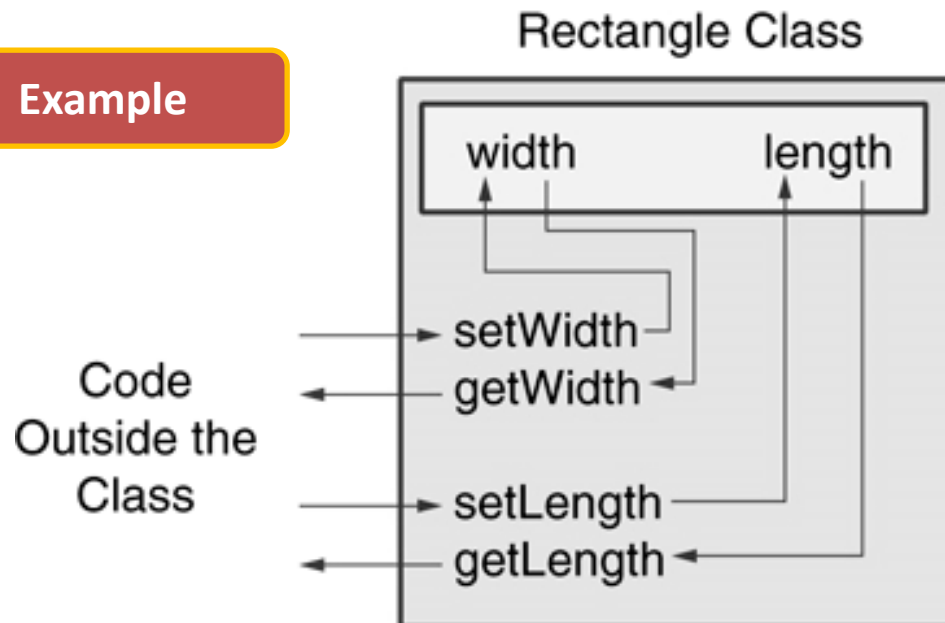
 Data can be accessed only through **public** functions

 Public functions define the **class's public interface**

Private Members : How to Access Private Members?

🌸 Code outside the class must use the class's public member functions (methods) to interact with the object

Example



Separating Class Specification from Implementation

Separating Class Specification from Implementation

CONCEPT :

✿ Usually class declarations are stored in their own header files.

✿ Member function definitions are stored in their own .cpp files

✿ A header file that contains a **class declaration** is called a **class specification file**.

✿ The name of the **class specification** file is usually the **same** as the **name of the class**, with a **.h extension**

Separating Class Specification from Implementation : example

- ✿ Place class declaration in a header file that serves as the **class specification file**. Name the file `ClassName.h`, for example, `Rectangle.h`
- ✿ Place member function definitions in `ClassName.cpp`, (called **implementation file**) for example, `Rectangle.cpp`
File should `#include` the class specification file.
- ✿ Programs that use the class ,(called **application file / driver prog.**) must `#include` the class specification file, and be compiled and linked with the member function definitions.

Contents of Rectangle.h

```
// Specification file for the Rectangle class.
#ifndef RECTANGLE_H
#define RECTANGLE_H

// Rectangle class declaration.
class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double);
        void setLength(double);
        double getWidth() const;
        double getLength() const;
        double getArea() const;
};
```

This directive tells the preprocessor to see if a constant named `RECTANGLE_H` has *not* been previously created with a `#define` directive

If the `RECTANGLE_H` constant has *not* been defined, these lines are included in the program. Otherwise, these lines are not included in the program


```
#ifndef RECTANGLE_H
#define RECTANGLE_H ←
class Rectangle
{
    // Member declarations
    // appear here.
};
#endif
```

The first included line defines the RECTANGLE_H constant. If this file is included again, the include guard will *skip* its contents

Contents of Rectangle.cpp

```
// Implementation file for the Rectangle class
```

```
#include "Rectangle.h" →
```

```
#include <iostream>
```

```
#include <cstdlib>
```

```
using namespace std;
```

```
// setWidth definition
```

```
// setLength definition
```

```
// getWidth definition
```

```
// getLength definition
```

```
// getArea definition
```

This directive includes the **Rectangle.h** file, which contains the **Rectangle class declaration**.

Main Program File

(Contents of useRectangle.cpp)

```
// This program should be compiled with Rectangle.h file, Rectangle.cpp file
#include "Rectangle.h"
#include <iostream>

using namespace std;
int main()

{
    Rectangle box;           //Define an instance
    double rectWidth;        //Local variable
    double rectLength;       //Local variable
    ..
    ...

}
```

Separating Class Specification from Implementation : Example

The implementation file `Rectangle.cpp`

The specification file `Rectangle.h`

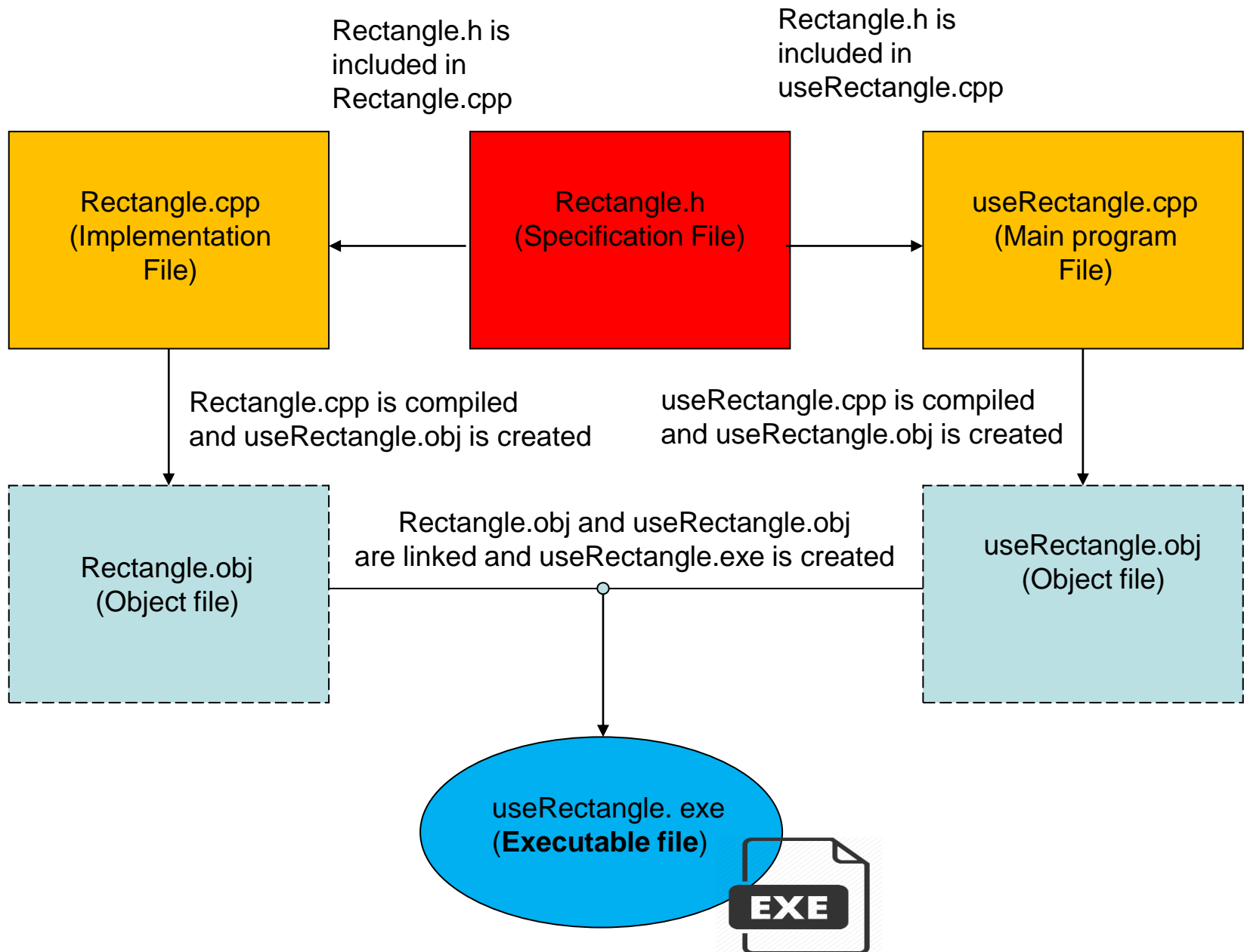
The main program `useRectangle.cpp`

 `Rectangle.obj` and `useRectangle.obj`

Create an executable program ...



`useRectangle.exe`



The process to create an executable program

Inline Member Functions

Inline Member Functions

✿ Member functions can be defined

- ◆ inline: in class declaration
- ◆ after the class declaration

✿ Inline appropriate for short function bodies:

Example

```
int getWidth() const  
{ return width; }
```

Rectangle Class with Inline Member Functions

```
class Rectangle
{
    private:
        double width;
        double length;
    public:
        void setWidth(double) ;
        void setLength(double) ;
        double getWidth() const
            { return width; }
        double getLength() const
            { return length; }
        double getArea() const
            { return width * length; }
};
```

**3 inline
member functions**

Tradeoffs – Inline vs. Regular Member Functions

✿ Regular functions – when called, compiler stores return address of call, **allocates memory for local variables**, etc.

✿ Code for an inline function is copied into program in place of call – **larger executable program**, but **no function call overhead**, hence **faster execution**