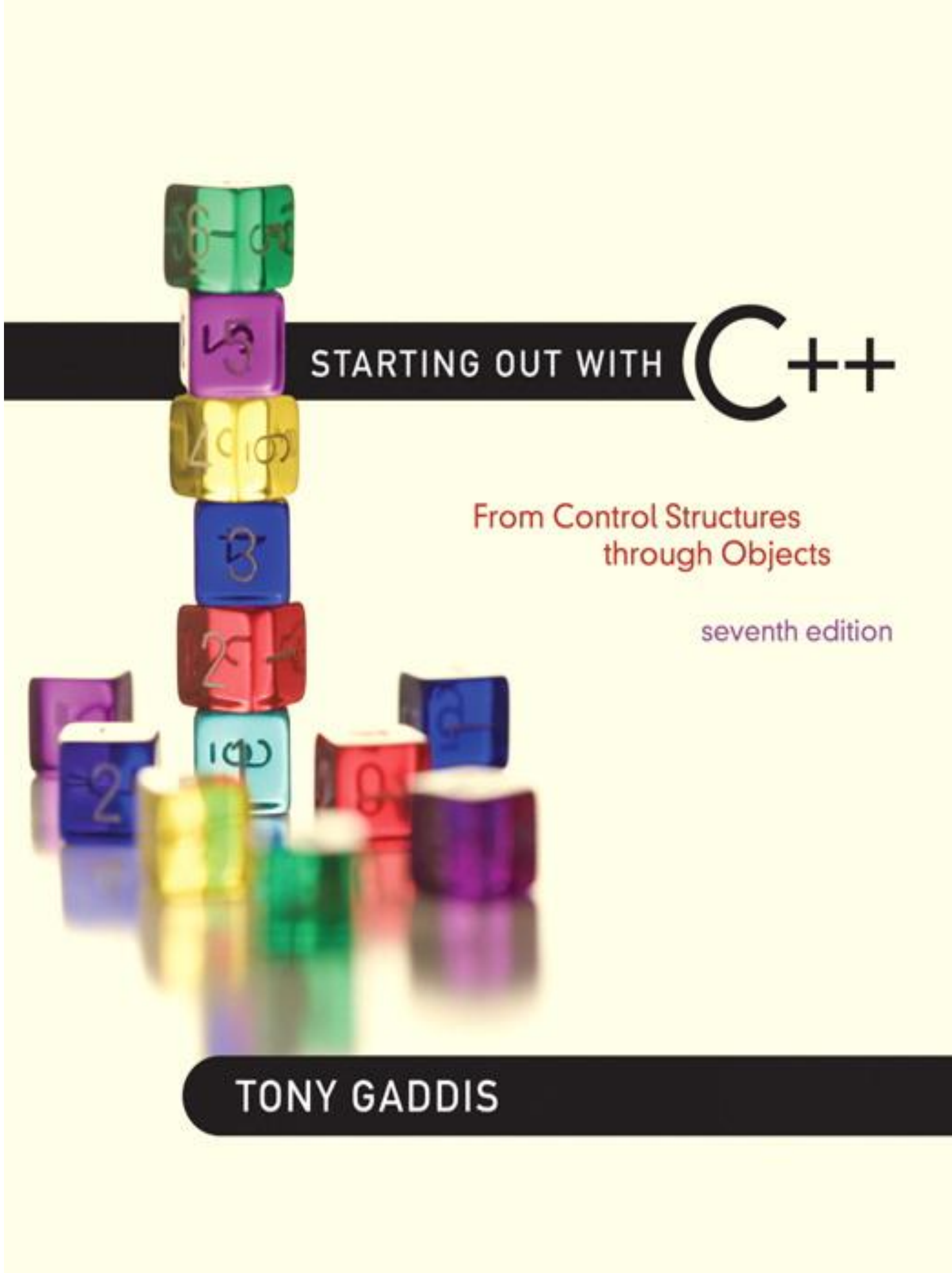
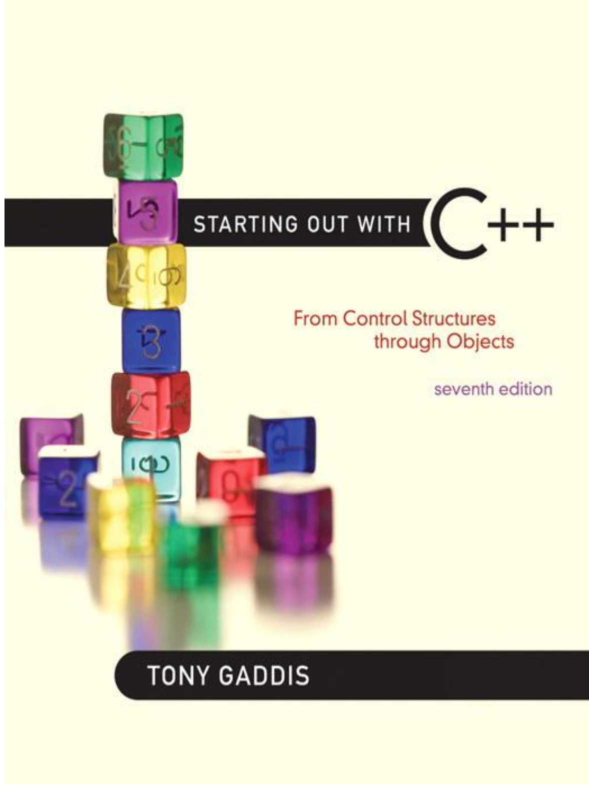
- Most of the content’s of this slide is taken from Tony Gadddies slide

Chapter 8:

Introduction to Classes





8.1

Procedural and Object-Oriented Programming

Procedural and Object-Oriented Programming

- Procedural programming focuses on the process/actions that occur in a program

- Object-Oriented programming is based on the data and the functions that operate on it. Objects are instances of ADTs that represent the data and its functions

Limitations of Procedural Programming

- If the data structures change, many functions must also be changed
- **Programs with Excessive Global Data**
- Programs that are based on complex function hierarchies are:
  - difficult to understand and maintain
  - difficult to modify and extend
  - easy to break

OOP

- Just as procedural programming is centered around functions, object-oriented programming is centered around *objects,* the abstract data types which package together both the data and the functions that operate on the data.

Object-Oriented Programming Terminology

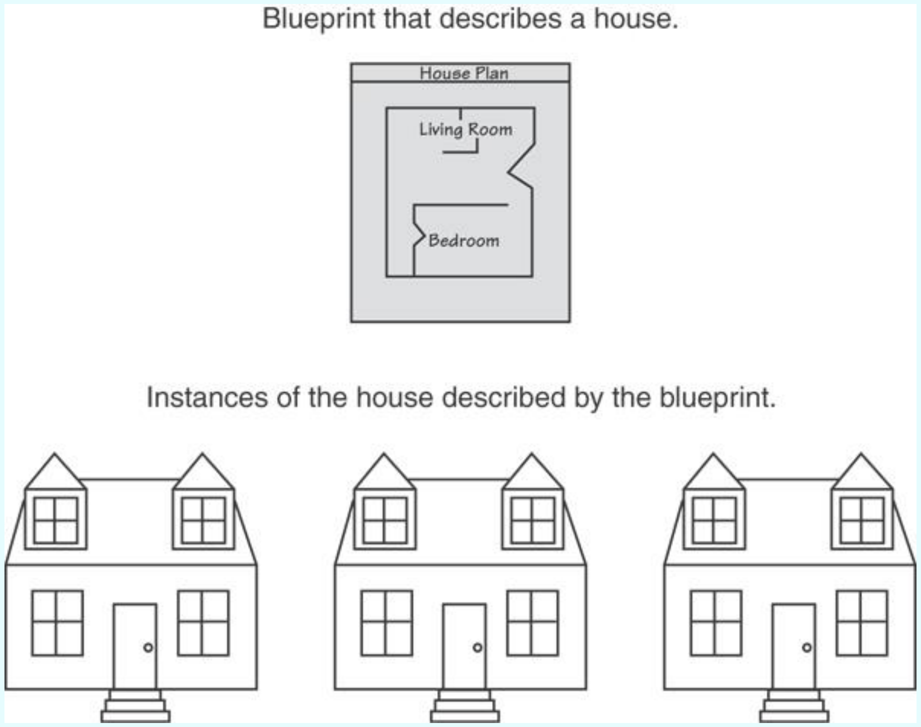
- class: like a `struct` (allows bundling of related variables), but variables and functions in the class can have different properties than in a `struct`.

- They are normally ADTs which encapsulate both data members and a set of functions that operate on them

- object: an instance of a `class`, in the same way that a variable can be an instance of a `struct`

Classes and Objects

- A Class is like a blueprint and objects are like houses built from the blueprint


Blueprint that describes a house.
House Plan
Living Room
Bedroom
Instances of the house described by the blueprint.

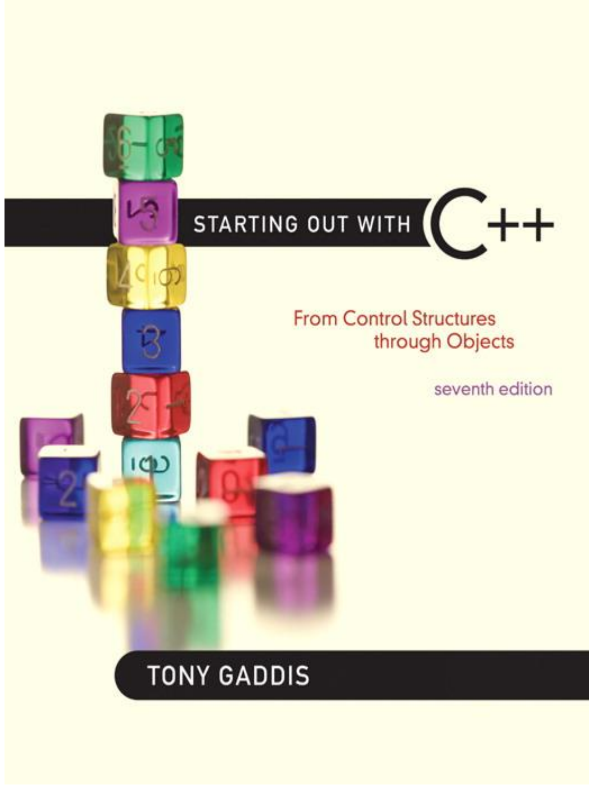
Object-Oriented Programming Terminology

- attributes: members variables of a class
- Aka fields
- methods or behaviors: member functions of a class

More on Objects

- data hiding: restricting access to certain members of an object

- public interface: members of an object that are available outside of the object. This allows the object to provide access to some data and functions without sharing its internal details and design, and provides some protection from data corruption



8.2

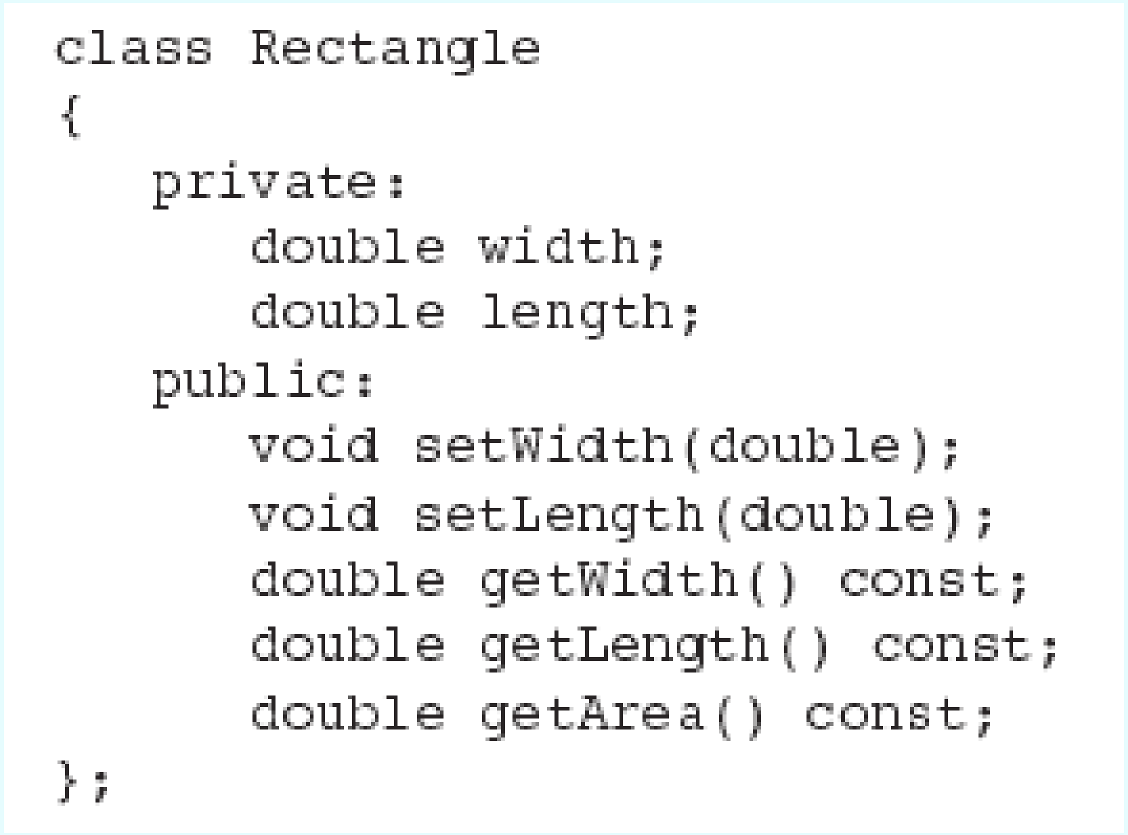
Introduction to Classes

Introduction to Classes

- Objects are created from a `class`
- Format:

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

Class Example

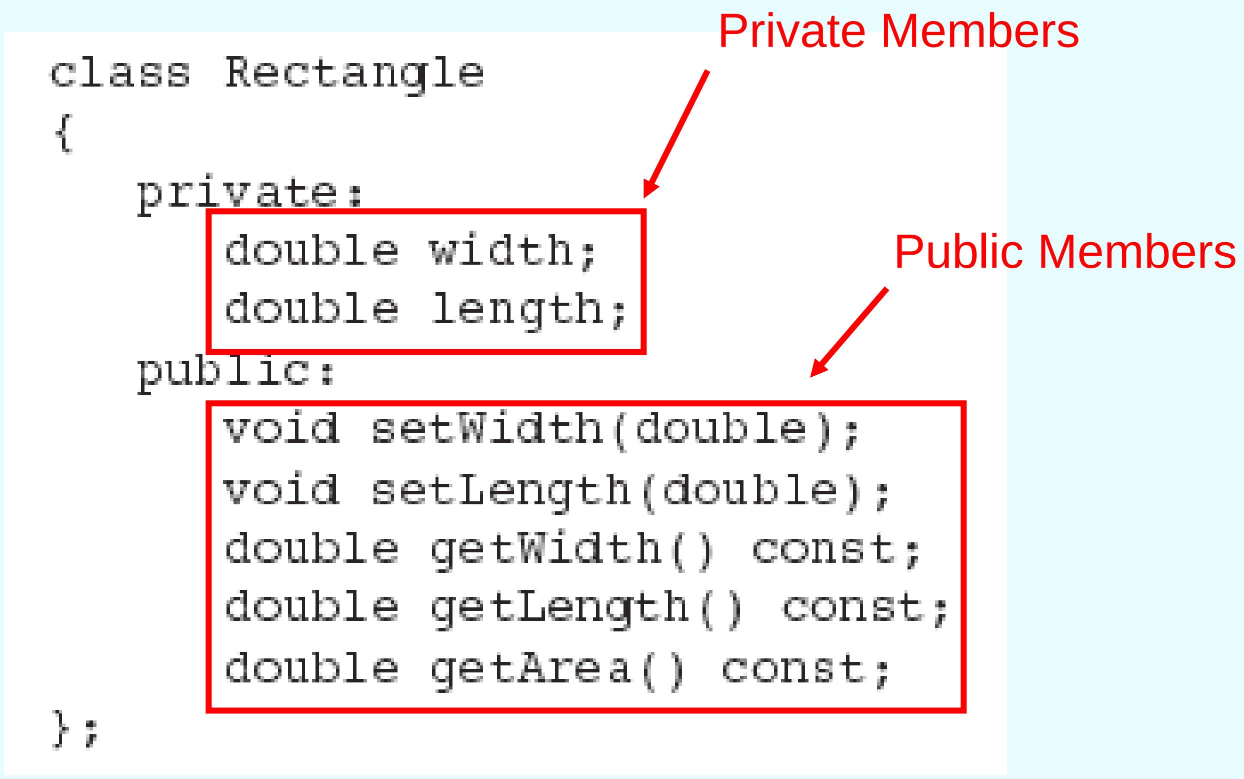


```
class Rectangle
{
   private:
      double width;
      double length;
   public:
      void setWidth(double);
      void setLength(double);
      double getWidth() const;
      double getLength() const;
      double getArea() const;
};
```

Access Specifiers

- In C++, there are three 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
- `Protected-`members cannot be accessed from outside the class, however, they can be accessed in inherited classes

Class Example



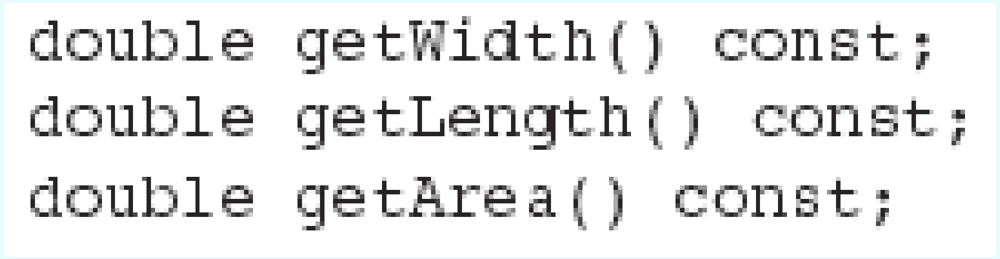
```
class Rectangle
{
   private:
      double width;
      double length;
   public:
      void setWidth(double);
      void setLength(double);
      double getWidth() const;
      double getLength() const;
      double getArea() const;
};
```

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`

Using `const` With Member Functions

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



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

Defining a Member Function

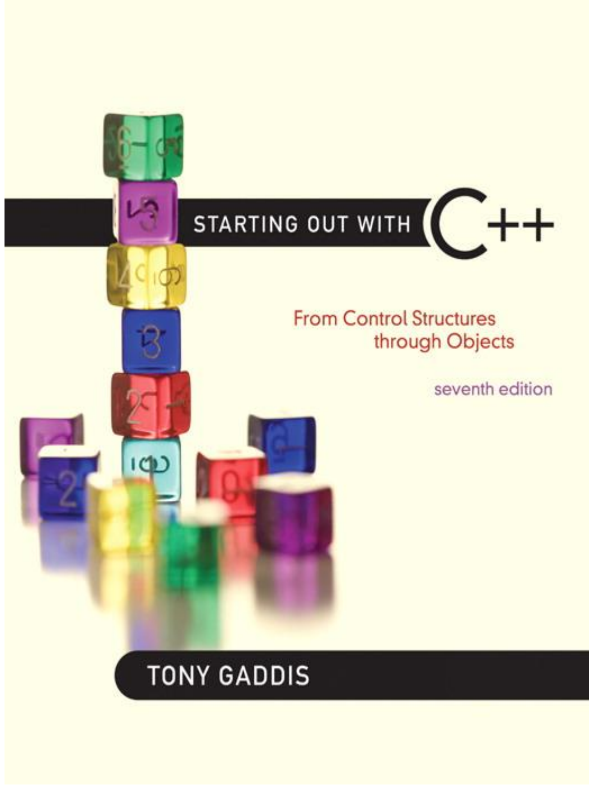
- When defining a member function:
  - Put prototype in class declaration
  - Define function using class name and scope resolution operator `(::)`

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

Accessors and Mutators

- Mutator: a member function that stores a value in a private member variable, 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 data, so they should be marked `const`.



8.3

Defining an Instance of a Class

Defining an Instance of a Class

- An object is an instance of a class
- 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

Program 13-1

```
// This program demonstrates a simple class.
#include <iostream>
using namespace std;

// 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;
};

//**************************************************
// setWidth assigns a value to the width member.   *
//**************************************************

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

//**************************************************
// setLength assigns a value to the length member. *
//**************************************************

```

Program 13-1 (Continued)

```
void Rectangle::setLength(double len)
{
   length = len;
}

//***************************************************
// getWidth returns the value in the width member. *
//***************************************************

double Rectangle::getWidth() const
{
   return width;
}

//****************************************************
// getLength returns the value in the length member. *
//****************************************************

double Rectangle::getLength() const
{
   return length;
}

```

Program 13-1 (Continued)

```
//*****************************************************
// getArea returns the product of width times length. *
//*****************************************************

double Rectangle::getArea() const
{
   return width * length;
}

//*****************************************************
// Function main                                      *
//*****************************************************

int main()
{
   Rectangle box;     // Define an instance of the Rectangle class
   double rectWidth;  // Local variable for width
   double rectLength; // Local variable for length

   // Get the rectangle's width and length from the user.
   cout << "This program will calculate the area of a\n";
   cout << "rectangle. What is the width? ";
   cin >> rectWidth;
   cout << "What is the length? ";
   cin >> rectLength;

   // Store the width and length of the rectangle
   // in the box object.
   box.setWidth(rectWidth);
   box.setLength(rectLength);
```

Program 13-1 (Continued)

```

    // Display the rectangle's data.
    cout << "Here is the rectangle's data:\n";
    cout << "Width: " << box.getWidth() << endl;
    cout << "Length: " << box.getLength() << endl;
    cout << "Area: " << box.getArea() << endl;
    return 0;
}
```

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
```

Excercise

- Create a class named 'Student' with a string variable 'name' and an integer variable 'roll_no'. Assign the value of roll_no as '2' and that of name as "John" by creating an object of the class Student.

Assignmnet

- Read about the following concepts and write a maximum of two pages of report
  - Encapsulation
  - Inheritance
  - Polymorphism
- Send before April 27 using the bot even if it doesn’t say document sent. When I receive, it will notify you.