CMSC202
Computer Science II for Majors

Lecture 06 –
Classes and Objects

Dr. Katherine Gibson

Last Class We Covered

- Pointers
  - Review
  - Passing to Functions
- Using pointers to pass arrays to functions
  - Including C-Strings
- References
  - Creating
  - Passing to Functions

Any Questions from Last Time?

Today's Objectives

- To understand the purpose and benefits of Object Oriented Programming

- To learn about classes in C++
  - Class Methods (Functions)

Programming and Abstraction

- All programming languages provide some form of ***abstraction***
  - Also called "information hiding"
  - Separates code use from code implementation
- Procedural Programming
  - Data Abstraction: using data structures
  - Control Abstraction: using functions
- Object Oriented Programming
  - Data and Control Abstraction: using ***classes***

Procedural vs OOP Examples

- Procedural
  - Calculate the area of a circle given the specified radius
  - Sort this class list given an array of students
  - Calculate the student's GPA given a list of courses

- Object Oriented
  - Circle, you know your radius, what is your area?
  - Class list, sort your students
  - Transcript, what is this student's GPA?

What is a Class?

- According to the dictionary:
  - A set, collection, group, or configuration containing members regarded as **having** certain **attributes or traits in common**

- According to OOP principles:
  - A group of objects with **similar properties**, **common behavior**, **common relationships** with other objects, and **common semantics**

Blueprints

- Classes are “blueprints” for creating objects
  - A dog class to create dog objects
  - A car class to create car objects
  - A shoe class to create shoe objects
- The blueprint defines
  - The class’s state/attributes
    - As variables
  - The class’s behaviors
    - As methods

Objects

- Each instance of a class is also called an ***object*** of that class type

- You can create as many instances of a class as you want
  - Just like a "regular" data type, like **`int`** or **`float`**
  - There is more than one dog, or car, or shoe

Encapsulation

- ***Encapsulation*** is a form of information hiding and abstraction
- Data and functions that act on that data are located in the same place (inside a class)
- Goal:
  - Separate interface from implementation so that someone can use the code without any knowledge of how it works

Class Declaration Example

```
class Car
{
   public:
      bool AddGas(float gallons);
      float GetMileage();
      // other operations

   private:
      float m_currGallons;
      float m_currMileage;
      // other data
};
```

Class Name

Protection Mechanism

Methods

Protection Mechanism

Data

Class Rules – Coding Standard

- Class names
  - Always begin with capital letter
  - Use mixed case for phrases
  - General word for class (type) of objects
    - Ex: Car, Boat, Building, DVD, List, Customer, BoxOfDVDs, …
- Class data (member variables)
  - Always begin with m_
    - Ex: m_fuel, m_title, m_name, …
- Class operations/methods
  - Always begin with capital letter
    - Ex: AddGas(), Accelerate(), ModifyTitle(), RemoveDVD(), …

Methods and Member Variables

- Classes ***encapsulate*** both data and functions
  - Class definitions must contain both
- Member variables are the data of a class
  - Its attributes, or characteristics
  - e.g., breed of Dog, size of Shoe, make of Car
- Class methods are used to act on that data
  - e.g., Play() with Dog, Inspect() a Car

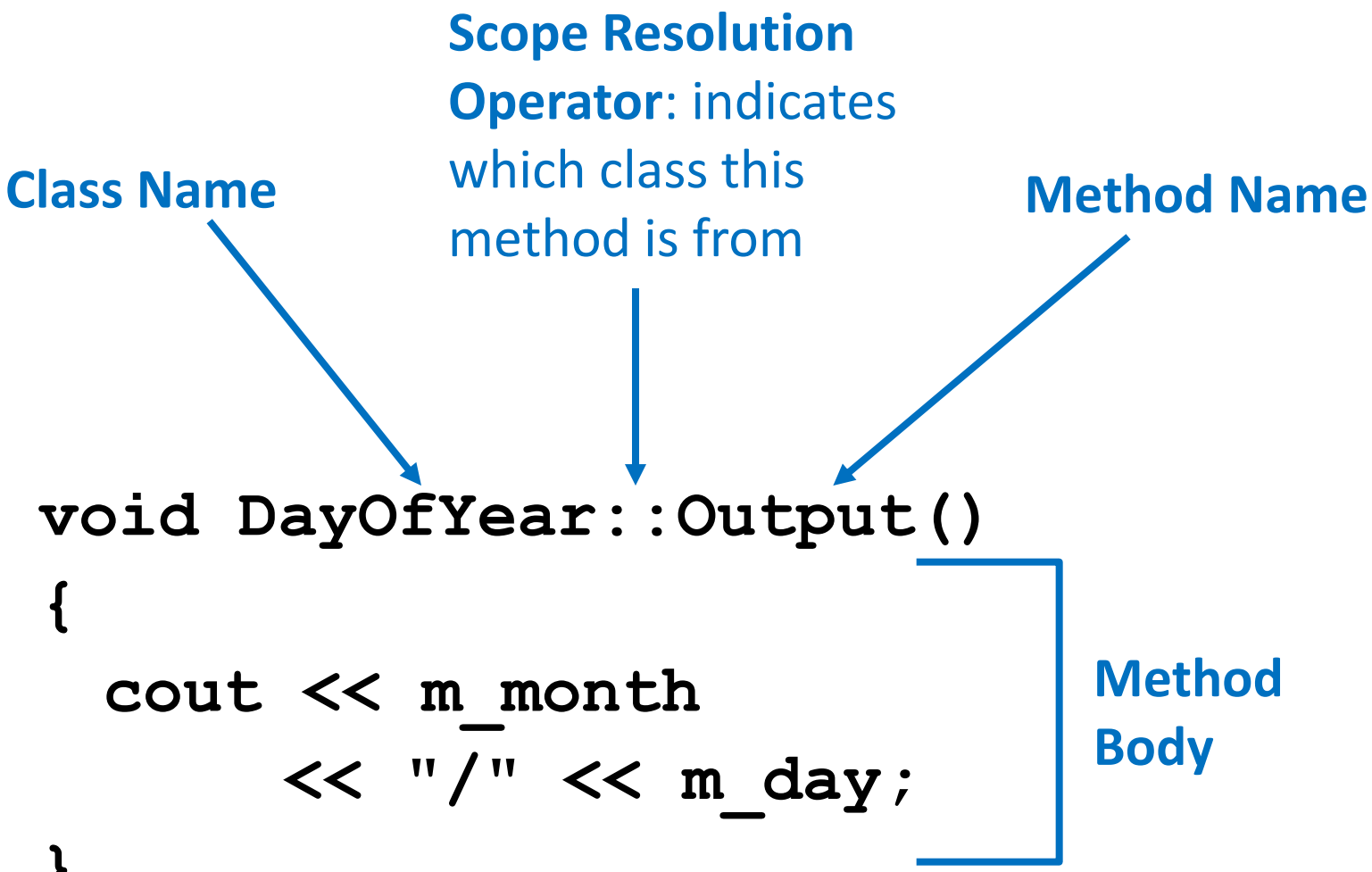
Example of Using a Class

```
// Represents a Day of the Year
class DayOfYear
{
    public:
        void Output();
        int m_month;
        int m_day;
};

// Output method - displays a DayOfYear
void DayOfYear::Output()
{
    cout << m_month << "/" << m_day;
}

// Code from main()
DayOfYear july4th;
july4th.m_month = 7;
july4th.m_day = 4;
july4th.Output();
```

Method Implementation



Separating Classes into Files

```
// Represents a Day of the Year
class DayOfYear
{
   public:
    void Output();
    int m_month;
    int m_day;
};

// Output method - displays a DayOfYear
void DayOfYear::Output()
{
   cout << m_month << "/" << m_day;
}
```

Class Declaration

Goes in file ClassName.h

(DayOfYear.h)

Class Definition

Goes in file ClassName.cpp

(DayOfYear.cpp)

Using Classes

```
// Code from main()
DayOfYear july4th;
july4th.m_month = 7;
july4th.m_day = 4;
july4th.Output();
```

Constructor (we'll cover this soon)

Object Name (Variable)

Dot Operator

Class Methods and Members

Dot and Scope Resolution Operator

- Used to specify "of what thing" they are members
- Dot operator:
  - Specifies member of particular object
    - Class method or member variable
- Scope resolution operator:
  - Specifies what class the function's definition belongs to

More about Encapsulation

- Class methods do not need to be passed information about that class object
  - Notice that the **`Output()`** method does not have any parameters
- Class methods are called ***on*** a class object
  - They know everything about that object already
- Remember, classes contain code and data!

Time for…

LIVECODING!!!

Livecoding Exercise

- Create a Rectangle class with
  - Member variables for height and width
  - Class methods to:
    - Calculate area
    - Calculate perimeter
    - Check if it's square
    - "Rotate" the rectangle
- Create both Rectangle.h and Rectangle.cpp

Announcements

- Project 1 has been released
- Found on Professor's Marron website
- Due by 9:00 PM on February 23rd
- Get started on it now!
- Make sure to read and follow the **coding standards** for this course!

- Next time: more on Classes and Objects