CMSC202 Computer Science II for Majors

Lecture 07 – Classes and Objects (Continued)

Dr. Katherine Gibson

Last Class We Covered

- Object Oriented Programming
  - Versus Procedural Programming
- Classes
  - Members
    - Member variables
    - Member functions (class methods)
- Livecoding: Rectangle class

Any Questions from Last Time?

Today’s Objectives

- To understand more about classes in C++
  - Learn the uses for access modifiers
  - Discuss more types of methods
    - Accessors
    - Mutators
    - Facilitators
    - Constructors
  - Overloading class methods

Class Access Specifiers

Access Specifiers

- In our definition for the **`DayOfYear`** class, everything was public
  - This is <u>not</u> good practice!

- Why?
  - Encapsulation! We don't want the end user to have direct access to the data
  - Why?
    - May set variables to invalid values

Access Specifier Types

- We have three different options for access specifiers, each with their own role:
  - **`public`**
  - **`private`**
  - **`protected`**

- Used to specify access for member variables and functions <u>inside</u> the class

Toy Syntax Example

```
class Date {
    public:
        int m_month;
    private:
        int m_day;
    protected:
        int m_year;
};
```

Public Access Specifier

- `public`
  - Anything that has access to a `Date` object also has access to all public member variables and functions
- Normally used for functions
  - But not all functions
- Need to have at least one public member
  - Why?

Private Access Specifier

- **`private`**
  - Private member variables and functions can only be accessed by member functions of the **`Date`** class
  - Cannot be accessed in **`main()`**, in other files, or by other functions

- If not specified, members default to private
- Should specify anyway – good coding practices!

Protected Access Specifier

- **`protected`**
  - Protected member variables and functions can only be accessed by:
    - Member functions of the **`Date`** class
    - Member functions of any derived classes

- (We'll cover this in detail later)

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Access Specifiers for Date Class

```
class Date {
   ???????:
      void Output();
   ???????:
      int m_month;
      int m_day;
      int m_year;
};
```

Access Specifiers for Date Class

```
class Date {
   public:
      void Output();
   private:
      int m_month;
      int m_day;
      int m_year;
};
```

Other Member Functions

New Member Functions

- Now that **`m_month`**, **`m_day`**, and **`m_year`** are private, how do we give them values, or retrieve those values?

- Write public member functions to provide indirect, controlled access for the user
- Remember, there is an ideal:
  - User only knows interface (public functions) not implementation (private variables)

Member Function Types

- There are many ways of classifying types, but here are the ones we'll use:

• Accessors	("Getters")
• Mutators	("Setters")
• Facilitators	("Helpers")

Member Function: Accessors

- Name starts with **`Get`**, ends with member name
- Allows retrieval of private data members

- Examples:

```
int GetMonth();
int GetDay();
int GetYear();
```

- Don’t generally take in arguments

Member Function: Mutators

- Name starts with **`Set`**, ends with member name
- Allows *controlled* changing of the value of a private data member

- Examples:

```
void SetMonth(int month);
void SetDay  (int day);
void SetYear (int year);
```

- Don't generally return anything

Mutator for SetMonth()

- How would you design a good mutator for the **`SetMonth()`** member function?

```
void Date::SetMonth(int month) {
    if (month >= 1 && month <= 12) {
        m_month = month;
    }
    else {
        m_month = 1; }
}
```

what's wrong with this function?

Better Mutator for SetMonth()

- This version of the **`SetMonth()`** member function doesn't use magic numbers!

```
void Date::SetMonth(int month) {
    if (month >= MIN_MONTH &&
        month <= MAX_MONTH) {
      m_month = month;
    } else {
      m_month = DEFAULT_MONTH; }
}
```

in what file would you store these constants?

Member Function: Facilitators

- Provide support for the class's operations
- `public` if generally called outside function
- `private/protected` if only called by member functions

- Examples:

```
void OutputMonth();            (public)
void IncrementDate();          (private)
```

Date with Specifiers

```
class Date {
public:
   void Output ();
   int  GetMonth();
   int  GetDay();
   int  GetYear();
   void SetMonth(int month);
   void SetDay  (int day);
   void SetYear (int year);
private:
   int m_month;
   int m_day;
   int m_year;
};
```

for the sake of brevity, we'll generally leave out the accessors and mutators when showing examples

Constructors

Constructors

- Special methods that "build" (construct) an object
  - Supply default values
  - Initialize an object

- Automatically called when an object is created
  - implicit: `Date today;`
  - explicit: `Date today(7, 28, 1914);`

Constructor Syntax

- Syntax:
  - For prototype:

```
ClassName();
```

  - For definition:

```
ClassName::ClassName() { /* code */ }
```

- Notice that…
  - There is no return type
  - Same name as class!

Constructor Definition

```
Date::Date (int month, int day,
            int year)
{
   m_month = month;
   m_day = day;
   m_year = year;
}
```

- What is missing from this constructor?
  - Technically, nothing -- but...
  - Validation of the information being passed in!

Better Constructor Definition

```
Date::Date (int month, int day,
            int year)
{
  if (m > 0 && m <= 12) {
    m_month = month; }
  else { m_month = 1; }
  if (d > 0 && d <= 31) {
    m_day = day; }
  else { m_day = 1; }
  if (y > 0 && y <= 2100) {
    m_year = year; }
  else { m_year = 1; }
}
```

is this the best way to handle this?

what might be a better solution?

Best Constructor Definition

```
Date::Date (int month, int day,
            int year)
{
    SetMonth(month);
    SetDay(day);
    SetYear(year);
}
```

- This allows us to reuse already written code

Time for…

LIVECODING!!!

Livecoding Exercise

- Update our Rectangle class with
  - Constructor
  - Accessors and Mutators
  - Class methods to:
    - Calculate area
    - Calculate perimeter
    - Check if it's Square
    - Print the rectangle's dimensions
- Create a **`main()`** function and use it!

Designing a Class

- Ask yourself:
  - What properties must each object have?
    - What data-types should each of these be?
    - Which should be private? Which should be public?
  - What operations must each object have?
    - What accessors, mutators, facilitators?
      - What parameters must each of these have?
        - » Const, by-value, by-reference, default?
      - What return value should each of these have?
        - » Const, by-value, by-reference?
    - Which should be private? Which should be public?
- Rules of thumb:
  - Data should be private (usually)
  - Operations should be public (usually)
  - At least 1 mutator and 1 accessor per data member (usually)

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: Wrap Up and Review for Exam 1!