Classes
Part 2

CMSC 202

Section Goals

- Abstraction
  - Provide a simple interface to other classes/functions
  - Information Hiding
    - Hide details of **data storage** and **implementation**
- Encapsulation
  - Control access to data
    - Private versus Public
- Definition…
  - Classes describe user-defined ADTs
    - ***Abstract Data Types***

Class Member Access

- Public
  - Any code can access this member
- Private
  - Only members of the class can access this member
- Default? If access mode unspecified, members are private

Syntax:

```
class ClassName
{
  public:
    // public functions
    // public data

  private:
    // private functions
    // private data
};
```

Improved DayOfYear Class

```
class DayOfYear
{
  public:
   void Input( );
   void Output( );
   void Set( int newMonth, int newDay );
   void Set( int newMonth );
   int GetMonthNumber( );
   int GetDay( );
  private:
   int m_month;
   int m_day;
};
```

This is the Class declaration – belongs in DayOfYear.h

Using DayOfYear Class

```
int main( )
{
   DayOfYear today;

   // Attempt to use private data…
   today.m_month = 2;           // ERROR!
   today.m_day = 23;            // ERROR!
   cout << "Today: " << m_month << "/"
        << m_day << endl;       // ERROR!

   // Instead, use public methods…
   today.Set( 2, 23 );
   cout << "Today: " << today.GetMonth() << "/"
        << today.GetDay() << endl;

   return 0;
}
```

Improved DayOfYear Class

```
class DayOfYear
{
  public:
   void Input( );
   void Output( );
   void Set( int newMonth, int newDay );
   void Set( int newMonth );
   int GetMonthNumber( );
   int GetDay( );
  private:
   int m_month;
   int m_day;
};
```

What are these methods?

Class Methods

Accessors
- Allow outside code to inspect a private data member
- Start with “Get” (usually)

Mutators
- Allow outside code to modify a private data member’
- Start with “Set” (usually)

Facilitators (Services)
- Provide some service for outside code
  - Print all class data
  - Retrieve data from user
  - Format data into a string
  - Calculate something

Accessors, Mutators, Facilitators?

```
class DayOfYear
{
   public:
    void Input( );
    void Output( );

    void Set( int newMonth, int newDay );
    void Set( int newMonth );

    int GetMonthNumber( );
    int GetDay( );
   private:
    int m_month;
    int m_day;
};
```

Facilitators

Mutators

Accessors

Class Implementation (Simple…)

```
void DayOfYear::Set( int newMonth, int newDay )
{
    m_month = newMonth;
    m_day = newDay;
}

void DayOfYear::Set( int newMonth )
{
    m_month = newMonth;
    m_day = 1;
}

int DayOfYear::GetMonthNumber( )
{
    return m_month;
}

int DayOfYear::GetDay( )
{
    return m_day;
}
```

These method implementations belong in DayOfYear.cpp file

How could the Set methods be improved?

Class Implementation (Improved)

```
//-----------------------------------------------
// Set
// PreConditions:
//        1 <= newMonth <= 12
//        1 <= newDay <= 31
// PostConditions:
//        day of year changed to user supplied values
//    if an error, exit program
//-----------------------------------------------
void DayOfYear::Set(int newMonth, int newDay)
{
    if ((newMonth >= 1) && (newMonth <= 12))
      m_month = newMonth;
    else
    {
      cout << "Illegal month value! Program aborted.\n";
      exit(1);
    }
    if ((newDay >= 1) && (newDay <= 31))
      m_day = newDay;
    else
    {
      cout << "Illegal day value! Program aborted.\n";
      exit(1);
    }
}
```

More Improvements

How else could this be improved?

- Valid day for each month
  - Ex: April has 30 days
- Valid day for month and year
  - Ex: February has 28 or 29 days, depending on year
- Bad data?
  - Set to “safe” value (ex: 1 for month or day)
  - Print an error & keep data
  - Return “false” to indicate illegal state
  - Set flag to “invalid object” (Zombie objects)

DayOfYear Input

```
void DayOfYear::Input( )
{
   cout << "Enter the month as a number: ";
   cin >> m_month;
   cout << "Enter the day of the month: ";
   cin >> m_day;

   if ((m_month < 1) || (m_month > 12)
       || (m_day < 1) || (m_day > 31))
   {
      cerr << "Illegal date! Program aborted.\n";
      exit(1);
   }
}
```

DayOfYear Output

```
void DayOfYear::Output( )
{
    switch (m_month)
    {
        case 1:  cout << "January   "; break;
        case 2:  cout << "February  "; break;
        case 3:  cout << "March     "; break;
        case 4:  cout << "April     "; break;
        case 5:  cout << "May       "; break;
        case 6:  cout << "June      "; break;
        case 7:  cout << "July      "; break;
        case 8:  cout << "August    "; break;
        case 9:  cout << "September "; break;
        case 10: cout << "October   "; break;
        case 11: cout << "November  "; break;
        case 12: cout << "December  "; break;
        default: cout << "Error in DayOfYear::Output."; break;
    }
    cout << m_day;
}
```

Using DayOfYear Class

```
int main( )
{
    DayOfYear today, bachBirthday;

    // input and echo today's date
    cout << "Enter today's date:\n";
    today.Input( );
    cout << "Today's date is  ";
    today.Output( ); cout << endl;

    // set and output JSB's birthday
    bachBirthday.Set(3, 21);
    cout << "J. S. Bach's birthday is  ";
    bachBirthday.Output( );
    cout << endl;
```

Using DayOfYear Class

```
    // CONT.
    // output special message
    if ((today.GetMonthNumber( ) == bachBirthday.GetMonthNumber( ))
        && (today.GetDay( ) == bachBirthday.GetDay( ) ))
      cout << "Happy Birthday Johann Sebastian!\n";
    else
      cout << "Happy Unbirthday Johann Sebastian!\n";
    return 0;
}
```

Class Design

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)

Guarding Header Files

To use a class, must #include declaration

```
#include "className.h"
```

Every file that uses class should #include it
- How do you protect from including twice?

```
#ifndef CLASSNAME_H
#define CLASSNAME_H
// class declaration here…
#endif
```

Guard EVERY .h file

Include EVERY .h file that you directly use