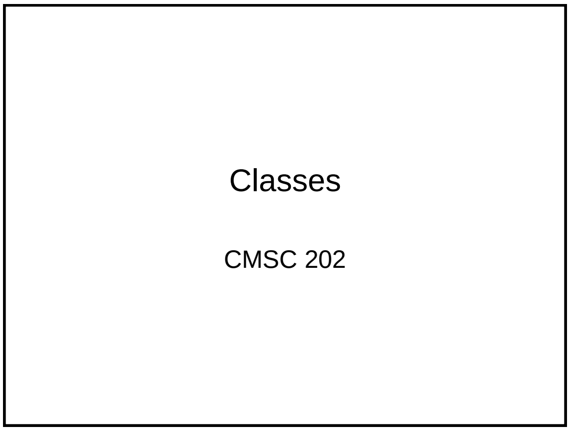
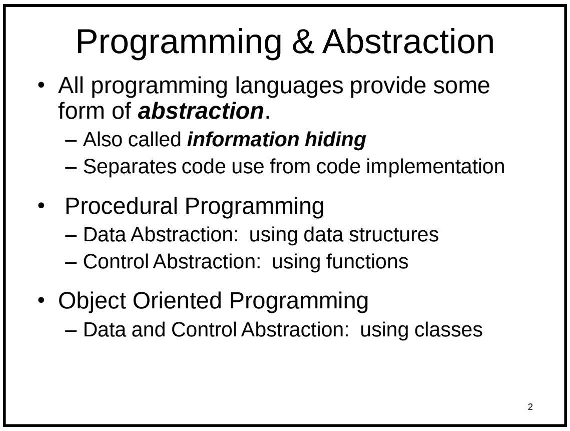


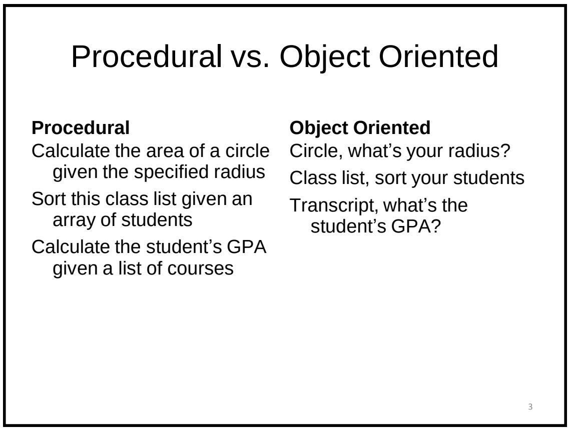
Classes

CMSC 202



Programming & 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. Object Oriented

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, what's your radius?

Class list, sort your students

Transcript, what's the student's GPA?

What is a Class?

- From the Dictionary
  - A kind or category
  - A set, collection, group, or configuration containing members regarded as ***having certain attributes or traits in common***
- From an Object Oriented Perspective
  - A group of objects with ***similar properties, common behavior, common relationships with other objects, and common semantics***
  - We use classes for ***abstraction*** purposes.

Classes

Classes are “blueprints” for creating a group of objects.

A bird class to create bird 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 behavior as methods

Class or Object?

- Variables of class types may be created just like variables of built-in types.
  - Using a set of blueprints you could create a bakery.
- You can create as many instances of the class type as you like.
  - There is more than one bakery in Baltimore.
- The challenge is to define classes and create objects that satisfy the problem.
  - Do we need an Oven class?

Structures

What about structs?

- Collection of data
- No operations explicitly related

```
struct DayOfYear
{
   int month;
   int day;
};

DayOfYear july4th;
july4th.month = 7;
july4th.day = 4;
```

Members

Structures

Good

- Simple
- Can be parameters to functions
- Can be returned by functions
- Can be used as members of other structs

Bad

- No operations
- Data is not protected
  - Any code that has access to the struct object has direct access to all members of that object

Classes – a Struct Replacement

Good

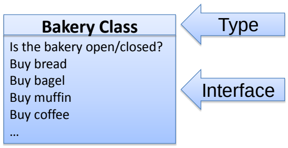
- Simple
- Objects can be parameters to functions
- Objects can be returned by functions
- Objects can be members of other classes
- Operations linked to data
- Data ***is*** protected
  - Code that uses an object MUST use the operators of the class to access/modify data of the object (usually)

Bad

- Nothing really…

Class Interface

- The requests you can make of an object are determined by its ***interface***.
- Do we need to know how bagels are made in order to buy one?
  - All we actually need to know is which bakery to go to and what action we want to perform.



Bakery Class

Is the bakery open/closed?
Buy bread
Buy bagel
Buy muffin
Buy coffee
…

Type

Interface

Implementation

Code and ***hidden data*** in the class that satisfies requests make up the class's ***implementation***.

What's hidden in a bakery?

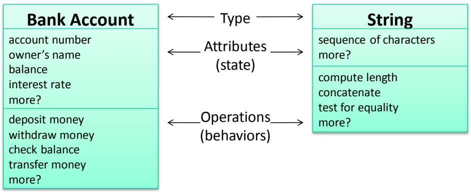
Every request made of an object must have an associated method that will be called.

In OO-speak we say that you are ***sending a message*** to the object, which responds to the message by executing the appropriate code.

Recall . . .

Class

- A ***complex data type*** containing:
  - Attributes – make up the object's ***state***
  - Operations – define the object's ***behaviors***



Bank Account

account number
owner's name
balance
interest rate
more?

deposit money
withdraw money
check balance
transfer money
more?

Type

Attributes
(state)

Operations
(behaviors)

String

sequence of characters
more?

compute length
concatenate
test for equality
more?

Class 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

Operations

Protection Mechanism

Data

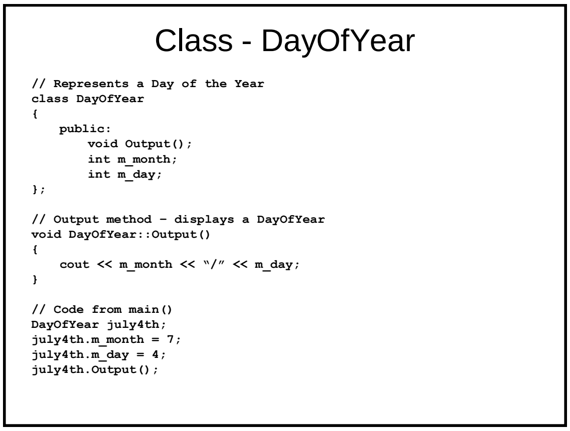
Struct vs. Class

```
struct DayOfYear
{
  int month;
  int day;
};

// Code from main()
DayOfYear july4th;
july4th.month = 7;
july4th.day = 4;
```

```
class DayOfYear
{
  public:
   int m_month;
   int m_day;
};

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

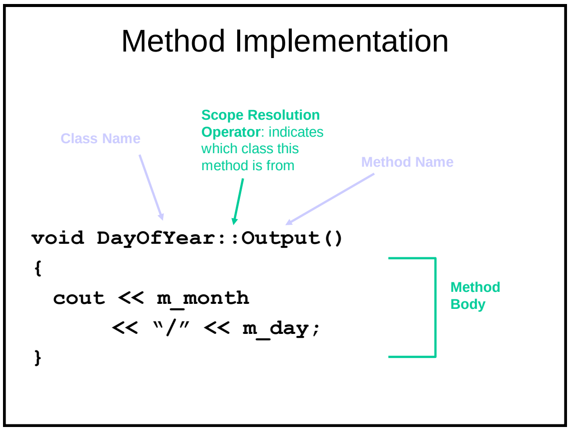
Class Rules – Coding Standard

- 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, CollectionOfRecords, …
- Class data
  - Always begin with m_
    - Ex: m_fuel, m_title, m_name, …
- Class operations/methods
  - Always begin with capital letter
    - Ex: AddGas(), Accelerate(), ModifyTitle(), RemoveDVD(), …



Class - DayOfYear

```
// 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

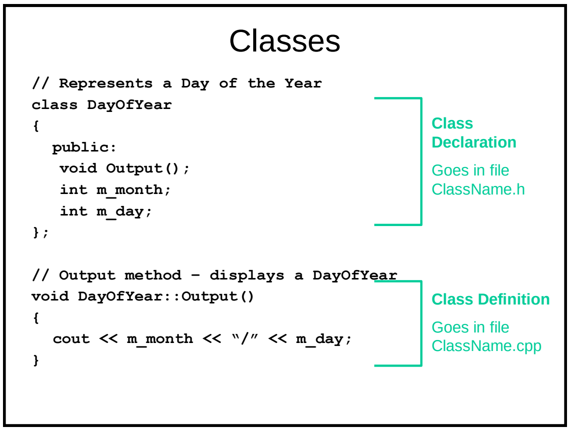
Class Name

Scope Resolution Operator: indicates which class this method is from

Method Name

```
void DayOfYear::Output()
{
  cout << m_month
       << “/” << m_day;
}
```

Method Body



Classes

```
// 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

Class Definition

Goes in file ClassName.cpp

Classes, Part II

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

Practice

- Design & Implement the "Stapler" class
  - Data
    - Number of Staples
      - Integer
      - Private
  - Operations
    - Fill – fill stapler to max capacity
      - Parameters? None
      - Return value? None
      - Public
    - Staple – dispense one staple
      - Parameters? None
      - Return value? Bool – was action successful or not
      - Public

Challenge

Design and Declare an “Alarm Clock” class that beeps when the alarm goes off…

- What properties?
- What operations?

Implement your Alarm Clock class

- Assume there are functions implemented in a standard library called:
  - int GetCurrentHour(); - returns 0 to 23
  - int GetCurrentMinute(); - returns 0 to 59
- Assume there exists an external mechanism to make the clock update every minute...keep it simple…

Write a main function that

- Displays the current time to the user
- Sets the alarm for 9:51 am (so that you’re not late for your 10 am class)

Classes, Part III

Warmup

Using the following **part** of a class, implement the Sharpen() method, it removes 1 from the length:

```
class Pencil
{
   public:
     bool Sharpen();
   private:
     int m_length;
};
```

Class Review

```
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;
};

// Declaring a DayOfYear object
DayOfYear today;
```

Facilitators

Mutators

Accessors

What's going on here?

Constructors

Special Methods that "build" (construct) an object

- Supply default values
- Initialize an object

Syntax:

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

Notice

- No return type
- Same name as class!

Constructor Example

```
class DayOfYear
{
   public:
    DayOfYear( int initMonth, int initDay );

    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;
};
```

Constructor Example Implementation

```
DayOfYear::DayOfYear( int initMonth, int initDay )
{
  m_month = initMonth;
  m_day = initDay;
}

// Improved version
DayOfYear::DayOfYear( int initMonth, int initDay )
{
  Set(initMonth, initDay);
}
```

How can this method be improved?

Why use a mutator?

Constructor Example Implementation

Initialization Lists

- Alternative to assignment statements (sometimes necessary!)
- Comma-separated list following colon in method definition

Syntax:

```
DayOfYear::DayOfYear( int initMonth, int initDay )
   : m_month( initMonth ), m_day( initDay )
{
}
```

Overloading Constructors

Yes – different parameter lists

Example

```
class DayOfYear
{
  public:
   DayOfYear( int initMonth, int initDay );

   DayOfYear( int initMonth );

   DayOfYear( );

   // other public methods…
  private:
   int m_month;
   int m_day;
};
```

Overloading Constructors

```
DayOfYear::DayOfYear( int initMonth, int initDay )
{
  Set(initMonth, initDay);
}

DayOfYear::DayOfYear( int initMonth )
{
  Set(initMonth, 1);
}

DayOfYear::DayOfYear( )
{
  Set(1, 1);
}
```

What would be another alternative to having all 3 of these methods?

Overloading Constructors

```
class DayOfYear
{
  public:
   DayOfYear( int initMonth = 1, int initDay = 1 );

   // other public methods…
  private:
   int m_month;
   int m_day;
};

DayOfYear::DayOfYear( int initMonth, int initDay  )
{
  Set(initMonth, initDay);
}
```

Default Parameters!

Constructors

- Why haven't we seen this before?
  - Compiler builds a default constructor
    - Unless you define a constructor…
- Think about the following:
  - vector<DayOfYear> days(20);
    - Calls default constructor for DayOfYear!
- What if something goes wrong?
  - One solution: Zombie objects
  - Another solution: Throw exception (later…)

Zombie Objects

```
class DayOfYear
{
   public:
    DayOfYear( int initMonth = 1, int initDay = 1 );

    bool isValid();

    // other public methods…
   private:
    int m_month;
    int m_day;
    bool m_isValid;
};

bool DayOfYear::isValid()
{
   return m_isValid;
}
```

```
DayOfYear::DayOfYear( int initMonth, int initDay )
   :m_month( initMonth), m_day( initDay )
{
   if (m_month < 1 || m_month > 12)
    m_isValid = false;
   else if ( m_day < 1 || m_day > 31)
    m_isValid = false;
   else if ( day too big for the specified month)
    m_isValid = false
   else
    m_isValid = true;
}
```

Practice

- Stapler class
  - What would the constructor look like?
    - Initialize a stapler to have 50 staples

Const and Objects

- With an Object

```
const DayOfYear jan1st(1, 1);
jan1st.Set(1, 5);     // ERROR
```

```
myfile.cpp: In function `int main()':
myfile.cpp:20: passing `const DayOfYear' as
  `this' argument of `void DayOfYear::Set(int,
  int)' discards qualifiers
```

Const and Methods

Const member functions

- Promise not to modify the current object
  - Usually accessors, print functions, …

Compiler checks

- Directly – is there an assignment to data member in method?
- Indirectly – is there a call to a non-const method?

Syntax

```
retType methodName(parameters) const;
```

Const Example

```
class DayOfYear
{
   public:
    DayOfYear( int initMonth = 1, int initDay = 1 );

    void Input( );
    void Output( ) const;

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

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

Promise not to alter data members!

Const Rules

Const member functions

- Can be called on const and non-const objects
- Can call other const member functions
- Cannot call non-const member functions

Non-const member functions

- Can be called only on non-const objects
  - Otherwise, compiler error!
- Can call const and non-const member functions

Const objects

- Can be passed as const argument

Non-const objects

- Can be passed as const or non-const argument

Practice?

What is wrong with this?

```
int DayOfYear::GetDay ( ) const
{
   if (m_day < 1 )
      Set( m_month, 1 );
   return m_day;
}
```

Practice

What is wrong with this?

```
void Bob ( const DayOfYear& doy)
{
   OutputDayOfYear ( doy );

   cout << "Please enter your birth month and day \n";

   int birthMonth, birthDay;
   cin >> birthMonth >> birthDay;

   doy.Set( birthMonth, birthDay );
}
```

Implementing with Const

Start from the beginning
- Don’t try to add const at the end of implementing

Use for
- Member functions that don’t change object
  - Facilitators (maybe) and Accessors (most definitely)
- Parameters whenever reasonable
  - Not with pass-by-value
  - Yes with pass-by-reference

Designing Classes

Ask yourself the following questions:

- What are the responsibilities of this type of object?
- What actions can an object take?
- What actions can another function take on an object?
- What information does an object store?
- What information does an object need access to?

For each method:

- What parameters (const, ref, const-ref, val)?
  - Preconditions – what values are legal for parameters?
- What return value (const, ref, const-ref, val)?
  - Postconditions – what was altered by method?
- Does this method change the object (const, non-const)?

Practice – Add const!

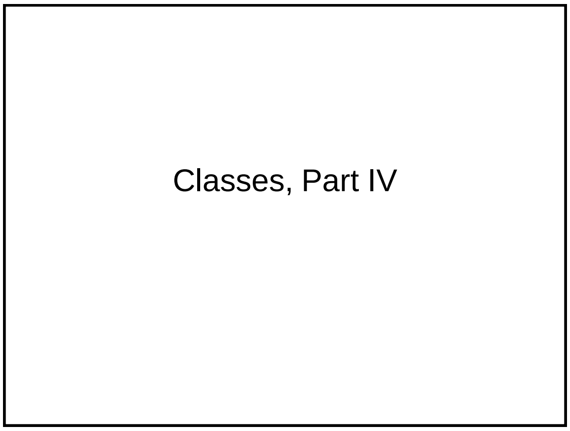
```
#include <string>
using namespace std;

class Person {
    public:
        Person( string name, int age );
        string GetName( );
        int GetAge( );
        void HappyBirthday( );
    private:
        string m_name;
        int m_age;
};
```

```
#include <iostream>
#include "Person.h"
using namespace std;

Person::Person( string name, int age )
{
    m_name = name;
    m_age = age;
}

string Person::GetName( )
{
    return m_name;
}

int Person::GetAge( )
{
    return m_age;
}

void Person::HappyBirthday( )
{
```

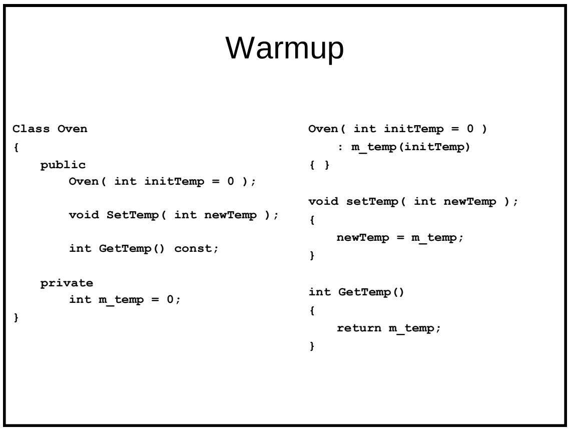
Challenge

Revisiting our Staple class

- Add a constructor
  - Initialize number of staples to the value of a parameter
- Retain the “Staple” method
  - Removes 1 staple
- Retain the “Fill” method
  - Completely fills to 100
- Add a “AddStaples” method
  - Adds some number of staples (parameter)
- Add a “GetNbrOfStaples” method
  - Returns the current number of Staples
- Add consts whenever appropriate
  - Parameters and methods!



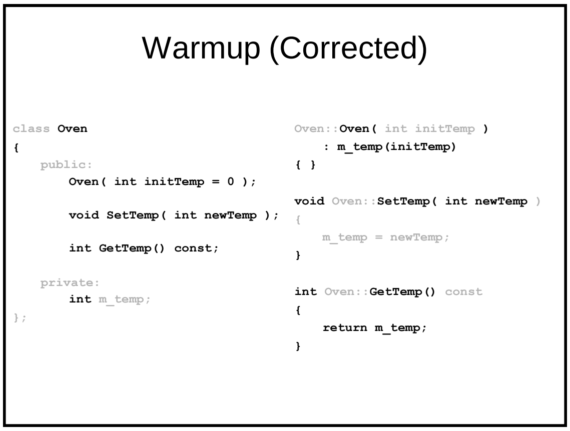
Classes, Part IV



Warmup

```
Class Oven
{
    public
        Oven( int initTemp = 0 );

        void SetTemp( int newTemp );

        int GetTemp() const;

    private
        int m_temp = 0;
}
```

```
Oven( int initTemp = 0 )
    : m_temp(initTemp)
{ }

void setTemp( int newTemp );
{
    newTemp = m_temp;
}

int GetTemp()
{
    return m_temp;
}
```



Warmup (Corrected)

```
class Oven
{
    public:
        Oven( int initTemp = 0 );

        void SetTemp( int newTemp );

        int GetTemp() const;

    private:
        int m_temp;
};
```

```
Oven::Oven( int initTemp )
    : m_temp(initTemp)
{ }

void Oven::SetTemp( int newTemp )
{
    m_temp = newTemp;
}

int Oven::GetTemp() const
{
    return m_temp;
}
```

Review

What term is used for “instance of a class”?
What is another term for “information hiding”?
What is a name for functions in a class?
What is a default constructor?
What are the limitations of a const object?
What does “const” mean with a method?

Student Class

Designing a Student…

- What data do we need?

Name
SSN
Address
Phone
Email ID
Course list
…

Let’s think about the Address, how can we represent that?

Aggregation

Objects can hold other objects!

- Class defines a private data member of another Class-type
- “has-a” relationship

Example

```
class Student
{
    public:
        // some methods…
    private:
        Address m_address;
        // more data…
};
```

Aggregation

We have 3 classes for this project

- MazeCell
- Maze
- MazeCrawler

How can we use aggregation here?

Aggregation – Another Look

```
class Vacation
{
    public:
        Vacation( int month, int day, int nbrOfDays );
        // more methods…
    private:
        DayOfYear m_startDay;
        int m_lengthOfTrip;
        // more data…
};

Vacation::Vacation( int month, int day, int nbrOfDays )
    : m_startDay(month, day), m_lengthOfTrip(nbrOfDays)
{
    // code…
}
```

What's going on here?

Implicit call to the Constructor! Remember – initializer lists were important! Only way to call Constructor!

Aggregation

```
class Vacation
{
   public:
      Vacation( int month, int day, int
        nbrOfDays );
      // more methods…
   private:
      DayOfYear m_startDay;
      int m_lengthOfTrip;
      // more data…
};
```

Can Vacation access DayOfYear's private data members?

Aggregation

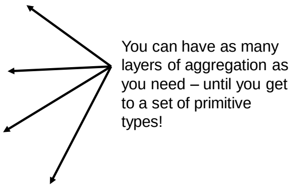
House “has-a”
- Front Door
- Set of bedrooms
- Garage
- Address

Garage “has-a”
- Lawnmower
- Rake
- Car

Car “has-a”
- Driver
- Set of passengers

Driver “has-a”
- Name
- Address

…



You can have as many layers of aggregation as you need – until you get to a set of primitive types!

Static

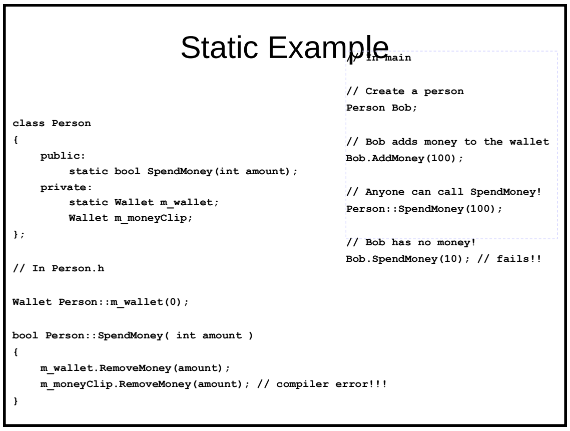
```
int foobar()          int foobar()
{                     {
   int a = 10;           static int a = 10;
   ++a;                  ++a;
   return a;             return a;
}                     }
```

Static and Classes?

Static data member
- ALL objects share data
- If one changes, affects all

Static methods
- Can access static data
- CANNOT access non-static data or methods

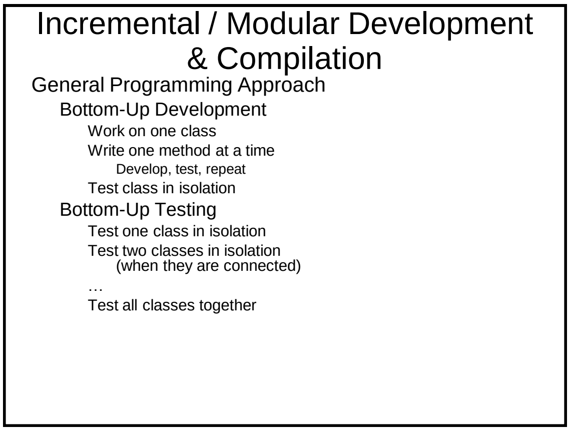
Regular methods
- Can access static data
- Can access non-static data and methods



Static Example

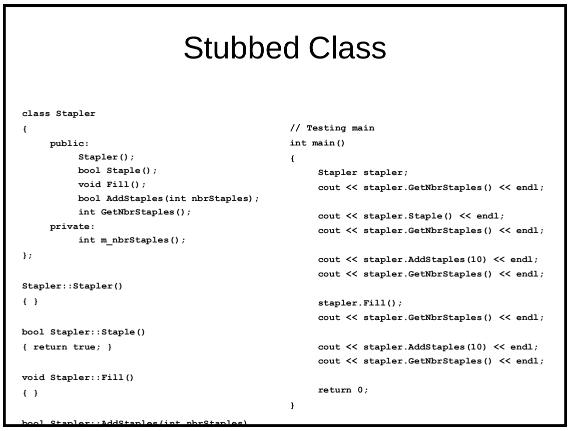
```
class Person
{
     public:
          static bool SpendMoney(int amount);
     private:
          static Wallet m_wallet;
          Wallet m_moneyClip;
};

// In Person.h

Wallet Person::m_wallet(0);

bool Person::SpendMoney( int amount )
{
     m_wallet.RemoveMoney(amount);
     m_moneyClip.RemoveMoney(amount); // compiler error!!!
}
```

```
// In main

// Create a person
Person Bob;

// Bob adds money to the wallet
Bob.AddMoney(100);

// Anyone can call SpendMoney!
Person::SpendMoney(100);

// Bob has no money!
Bob.SpendMoney(10); // fails!!
```



Incremental / Modular Development & Compilation

- General Programming Approach
  - Bottom-Up Development
    - Work on one class
    - Write one method at a time
      - Develop, test, repeat
    - Test class in isolation
  - Bottom-Up Testing
    - Test one class in isolation
    - Test two classes in isolation
      (when they are connected)
    - …
    - Test all classes together



Stubbed Class

```
class Stapler
{
     public:
          Stapler();
          bool Staple();
          void Fill();
          bool AddStaples(int nbrStaples);
          int GetNbrStaples();
     private:
          int m_nbrStaples();
};

Stapler::Stapler()
{ }

bool Stapler::Staple()
{ return true; }

void Stapler::Fill()
{ }

bool Stapler::AddStaples(int nbrStaples)
```

```
// Testing main
int main()
{
     Stapler stapler;
     cout << stapler.GetNbrStaples() << endl;

     cout << stapler.Staple() << endl;
     cout << stapler.GetNbrStaples() << endl;

     cout << stapler.AddStaples(10) << endl;
     cout << stapler.GetNbrStaples() << endl;

     stapler.Fill();
     cout << stapler.GetNbrStaples() << endl;

     cout << stapler.AddStaples(10) << endl;
     cout << stapler.GetNbrStaples() << endl;

     return 0;
}
```

P2 - Design

- Test cases
  - Use these with your Testing main
  - Run tests on your class EVERY time you modify it
- Implementation
  - Write 5 lines
  - Save
  - Compile
  - Test
  - Repeat

Challenge

- Come up with 1 GOOD example for each of the following:
  - Class that uses aggregation
  - Class that uses static data
    - This one may be tough…

- Do not use examples from class, slides, text, or lecture notes…