

Classes  
Part 1  
  
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

2

---

---

---

---

---

---

---

---

**Procedural vs. Object Oriented**

<p><b>Procedural</b> 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</p>	<p><b>Object Oriented</b> Circle, what's your radius? Class list, sort your students Transcript, what's the student's GPA?</p>
--	--

3

---

---

---

---

---

---

---

---

## 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.

4

---

---

---

---

---

---

---

---

## 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

5

---

---

---

---

---

---

---

---

## 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?

6

---

---

---

---

---

---

---

---

## Structures

2<sup>nd</sup> collection data type: structures (`struct`)

Structure: aggregate of values of different types

Compare to array: collection of values of same type

Treated as a single item, like arrays

Must first define struct before declaring any variables.

---

---

---

---

---

---

---

---

## Structure Types

Define struct globally (typically)

No memory is allocated

Just a placeholder for what our structure will look like

Example definition:

```
struct CDAccountV1 ← Name of new struct
{
    double balance; ← member names
    double interestRate;
    int term;
};
```

---

---

---

---

---

---

---

---

## Declare Structure Variable

With structure type defined, now declare variables of this new type:

```
CDAccountV1 account;
```

Just like declaring simple types

Variable `account` now of type `CDAccountV1`

Dot operator to access *member variables*:

```
account.balance
account.interestRate
account.term
```

---

---

---

---

---

---

---

---

## Structure Example: Display 6.1 A Structure Definition (1 of 3)

Display 6.1 A Structure Definition

```
1 //Program to demonstrate the CDAccountV1 structure type.
2 #include <iostream>
3 using namespace std;
4 //Structure for a bank certificate of deposit:
5 struct CDAccountV1
6 {
7     double balance;
8     double interestRate;
9     int term;//months until maturity
10 };
11 void getData(CDAccountV1& theAccount);
12 //Postcondition: theAccount.balance, theAccount.interestRate, and
13 //theAccount.term have been given values that the user entered at the keyboard
```

*An improved version of this structure will be given later in this chapter.*

---

---

---

---

---

---

---

---

## Structure Example: Display 6.1 A Structure Definition (2 of 3)

```
14 int main()
15 {
16     CDAccountV1 account;
17     getData(account);
18
19     double rateFraction, interest;
20     rateFraction = account.interestRate/100.0;
21     interest = account.balance*(rateFraction*(account.term/12.0));
22     account.balance = account.balance + interest;
23
24     cout.setf(ios::fixed);
25     cout.setf(ios::showpoint);
26     cout.precision(2);
27     cout << "When your CD matures in "
28         << account.term << " months,\n"
29         << "it will have a balance of $"
30         << account.balance << endl;
31
32     return 0;
33 }
```

(continued)

---

---

---

---

---

---

---

---

## Structure Example: Display 6.1 A Structure Definition (3 of 3)

Display 6.1 A Structure Definition

```
31 //Uses iostream:
32 void getData(CDAccountV1& theAccount)
33 {
34     cout << "Enter account balance: $";
35     cin >> theAccount.balance;
36     cout << "Enter account interest rate: ";
37     cin >> theAccount.interestRate;
38     cout << "Enter the number of months until maturity: ";
39     cin >> theAccount.term;
40 }
```

### SAMPLE DIALOGUE

```
Enter account balance: $100.00
Enter account interest rate: 10.0
Enter the number of months until maturity: 6
When your CD matures in 6 months,
it will have a balance of $105.00
```

---

---

---

---

---

---

---

---

## 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...

---

---

---

---

---

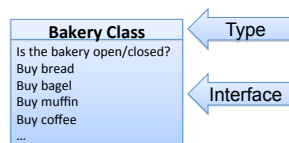
---

---

---

## 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.



---

---

---

---

---

---

---

---

## 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.

19

---

---

---

---

---

---

---

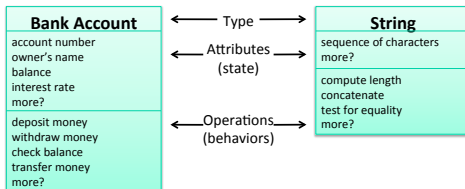
---

## Recall . . .

### Class

– A **complex data type** containing:

- Attributes – make up the object's **state**
- Operations – define the object's **behaviors**



20

---

---

---

---

---

---

---

---

## Class Example

```
class Car ← Class-name
{
  public: ← Protection Mechanism
    bool AddGas(float gallons);
    float GetMileage();
    // other operations } Operations
  private: ← Protection Mechanism
    float m_currGallons;
    float m_currMileage;
    // other data } Data
};
```

---

---

---

---

---

---

---

---

## Struct vs. Class

```
struct DayOfYear      class DayOfYear
{
    int month;        public:
    int day;          int m_month;
};                   int m_day;
                    };
// Code from main() // Code from main()
DayOfYear july4th;   DayOfYear july4th;
july4th.month = 7;   july4th.m_month = 7;
july4th.day = 4;     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;
};
```

Class Declaration  
Goes in file  
ClassName.h

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

Class Definition  
Goes in file  
ClassName.cpp

---

---

---

---

---

---

---

---

## Dot and Scope Resolution Operator

Used to specify "of what thing" they are members

Dot operator:

Specifies member of particular object

Scope resolution operator:

Specifies what class the function definition comes from

---

---

---

---

---

---

---

---



## A Class's Place

Class is full-fledged type!

Just like data types int, double, etc.

Can have variables of a class type

We simply call them "objects"

Can have parameters of a class type

Pass-by-value

Pass-by-reference

Can use class type like any other type!

6-28

---

---

---

---

---

---

---

---

## Encapsulation

Any data type includes

Data (range of data)

Operations (that can be performed on data)

Example:

*int* data type has:

Data: -2147483648 to 2147483647 (for 32 bit int)

Operations: +, -, \*, /, %, logical, etc.

Same with classes

But WE specify data, and the operations to be allowed on our data!

6-29

---

---

---

---

---

---

---

---

## Abstract Data Types

"Abstract"

Programmers don't know details

Abbreviated "ADT"

Collection of data values together with set of basic operations defined for the values

ADT's often "language-independent"

We implement ADT's in C++ with classes

C++ class "defines" the ADT

Other languages implement ADT's as well

6-30

---

---

---

---

---

---

---

---

## More Encapsulation

### Encapsulation

Means "bringing together as one"

Declare a class → get an object

Object is "encapsulation" of

Data values

Operations on the data (member functions)

---

---

---

---

---

---

---