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

2nd 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:

```c
struct CDAccountV1 { 
    double balance;  // member names
    double interestRate;
    int term;
};
```

Declare Structure Variable

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

```c
CDAccountV1 account;
```
Just like declaring simple types
Variable account now of type CDAccountV1
Dot operator to access member variables:

```c
account.balance
account.interestRate
account.term
```
### Display 6.1: A Structure Definition (1 of 3)

```c
// A Program to demonstrate the CDAccount structure type.

#include <iostream>

using namespace std;

// Structure for a bank certificate of deposit (CD) account:

struct CDAccount:
  double balance;
  double interestRate;
  int term; // months until maturity

void getdata(CDAccount& theAccount) {
  // Get the account balance, theAccount.interestRate, and
  // theAccount.term
  // Display user input for CD account
}
```

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

```c
// Description: Display the account balance, theAccount.interestRate, and
// theAccount.term have been given values that the user entered at the keypad
```

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

```c
void putdata(CDAccount theAccount)
// Display output on the screen:
```

**Announcement**

- Enter account balance: $100.00
- Enter interest rate: 5.0%
- Enter the number of months until maturity: 6
- Your CD matures in 6 months.
- It will have a balance of $110.00.
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.

---

Recall . . .

**Class**

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

<table>
<thead>
<tr>
<th>Bank Account</th>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>account number</td>
<td>Attributes (state)</td>
<td>sequence of characters more?</td>
</tr>
<tr>
<td>owner’s name</td>
<td></td>
<td>compute length</td>
</tr>
<tr>
<td>balance</td>
<td></td>
<td>compare to text for equality</td>
</tr>
<tr>
<td>interest rate</td>
<td></td>
<td>more?</td>
</tr>
<tr>
<td>more?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit money</td>
<td></td>
<td></td>
</tr>
<tr>
<td>withdraw money</td>
<td></td>
<td></td>
</tr>
<tr>
<td>check balance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transfer money</td>
<td></td>
<td></td>
</tr>
<tr>
<td>more?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Class Example

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

private:
    float m_currGallons;
    float m_currMileage;
    // other data
};
```
Struct vs. Class

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

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

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

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

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;
}
```

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!

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!

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
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)