CS 52 : C++ Programming

Howard A. Stahl
Agenda

• Defining Classes
• Members, Methods And Scope Resolution
• Delivering Classes
• Constructors
• Enumerations
• Default Parameter Values
An Object Has...

• State Described Via Attributes
  – every car has a make and a model

• Behavior Described Via Methods
  – every car can honk its horn

• Identity Described Via Instances
  – from the sea of all Honda Preludes, I can identify the one that is mine
Example: Bank Account

<table>
<thead>
<tr>
<th>bankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>void withdraw(double)</td>
</tr>
<tr>
<td>void deposit(double)</td>
</tr>
<tr>
<td>double balance()</td>
</tr>
<tr>
<td>string name()</td>
</tr>
<tr>
<td>void setName(string)</td>
</tr>
<tr>
<td>string my_name</td>
</tr>
<tr>
<td>double my_balance</td>
</tr>
</tbody>
</table>
Object-Oriented Programming

• The Power Of C++ Comes From Defining Custom Classes
  – abstract data types
### Example: Bank Account

<table>
<thead>
<tr>
<th>bankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>void withdraw(double)</td>
</tr>
<tr>
<td>void deposit(double)</td>
</tr>
<tr>
<td>double balance()</td>
</tr>
<tr>
<td>string name()</td>
</tr>
<tr>
<td>void setName(string)</td>
</tr>
</tbody>
</table>

| string my_name |
| double my_balance |

Header File (.h)

Implementation File (.cpp)

should have matching filenames
Header File

• The Header File Contains The Class Interface
  – what it does
  – NOT how it does it

• The Header File Is What Is Referenced By #include
  – custom classes use double quotes
  – system classes use < >
class bankAccount {

public:

//bunch of stuff goes here

private:

//bunch of stuff goes here

};
Baccount.h

class bankAccount {
    public:
        //bunch of stuff goes here
    
    private:
        //bunch of stuff goes here

};

public features of a class are available for use by consumers
Baccount.h

class bankAccount {
    public:
    //bunch of stuff goes here

    private:
    //bunch of stuff goes here

};

public features of a class are available for use by consumers

private features of a class are available for use by the class itself
Consider My Car

PROPERTIES
Make: Honda
Model: Prelude

FUNCTIONALITY
play_music
toggle_left_blinker
honk
Consider My Car

As a consumer, I get a very simplified interface and I don’t care much for any of the details...

**PROPERTIES**
- Make: Honda
- Model: Prelude

**FUNCTIONALITY**
- play_music
- toggle_left_blinker
- honk
Truth In Advertising

• public And private Keyword Can Be Listed Many Times In One Class Header File
• If Left Unstated, Its private
• Always A Good Idea To Just Have One Of Each
What Kind Of Stuff Can Be public Or private?

• Recall That Objects Have State And Behavior

• Classes Have Members And Methods
  – members are state variables held in each instance --- each instance has its own individual copy of the members
  – methods are functions that can be executed via dot notation by any instance
Example: Bank Account

```cpp
class bankAccount {
public:
    // bunch of stuff goes here

private:
    string my_name;
    double my_balance;

    void setName( string )
};
```
Use Of private Declarations

• Generally, All Member Variables Are Declared private
  – why?

• Generally, Any Helper Functions Are Declared private
  – why?
## Example: Bank Account

```cpp
class bankAccount {
public:
    void withdraw(double d);
    void deposit(double d);
    string name();
    void setName(string s);

private:
    string my_name;
    double my_balance;
};
```

<table>
<thead>
<tr>
<th>bankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>void withdraw(double)</td>
</tr>
<tr>
<td>void deposit(double)</td>
</tr>
<tr>
<td>double balance()</td>
</tr>
<tr>
<td>string name()</td>
</tr>
<tr>
<td>void setName(string)</td>
</tr>
</tbody>
</table>

String `my_name` and double `my_balance`
Use Of public Declarations

• Generally, Accessors And Mutators Are Made public
  – why?
Implementation File

- Header File Describes A Contract
- Implementation File Must Fulfill All The Header Declarations
How Are Class Methods Defined?

- C++ Provides Syntax To Separate Normal Functions From Class Functions
- Class Function Implementations Must Use Scope Resolution Operator
Example: Bank Account

<table>
<thead>
<tr>
<th>bankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>void withdraw( double )</td>
</tr>
<tr>
<td>void deposit( double )</td>
</tr>
<tr>
<td>double balance( )</td>
</tr>
<tr>
<td>string name( )</td>
</tr>
<tr>
<td>void setName( string )</td>
</tr>
<tr>
<td>string  my_name</td>
</tr>
<tr>
<td>double  my_balance</td>
</tr>
</tbody>
</table>

double bankAccount::balance(){
  return my_balance;
}

void bankAccount::deposit( double amt ) { 
  my_balance += amt;
}

void bankAccount::withdraw( double amt ) { 
  my_balance -= amt;
}
Example: Bank Account

class bankAccount
{
    string name();
    double balance();
    void deposit(double amt);
    void withdraw(double amt);
    void setName(string);

private:
    string my_name;
    double my_balance;
};

double bankAccount::balance()
{
    return my_balance;
}

void bankAccount::deposit(double amt)
{
    my_balance += amt;
}

void bankAccount::withdraw(double amt)
{
    my_balance -= amt;
}

Class Methods Have Access To Private Things
Delivering Classes

someClass

Header File (.h)  Implementation File (.cpp)

Object File (.obj)
Delivering Classes

The consumer of a class only needs the API (.h) and the .obj file for its link step.
Time For Our First Demo!

- Student.cpp

(See Handout For Example 1)
Summarizing Our First Demo!

• Consuming A Well-Written Class Is Easy!
• Creating A Well-Written Class Is Hard!
Instantiation

• Now That We Can Make Classes With State And Behavior, Making Instances Seems Harder Than It Looks At First
  – how do we set the default values of our member variables?
• We Need An “Entry Point” When Instances Are Made
• These Hooks Are Called “Constructors”
Constructors

- Constructors Are Called When Instances Are Declared
  - these methods are not programmer-callable!
- You Can Pass Parameters To Constructors
## Example: Bank Account

<table>
<thead>
<tr>
<th>bankAccount</th>
<th>bankAccount::bankAccount()</th>
</tr>
</thead>
<tbody>
<tr>
<td>void withdraw( double )</td>
<td>my_balance = 0.0;</td>
</tr>
<tr>
<td>void deposit( double )</td>
<td>my_name = &quot;?NAME?&quot;;</td>
</tr>
<tr>
<td>double balance( )</td>
<td>}</td>
</tr>
<tr>
<td>string name( )</td>
<td>bankAccount::bankAccount(</td>
</tr>
<tr>
<td>void setName( string )</td>
<td>string name,</td>
</tr>
<tr>
<td>string my_name</td>
<td>double amt ) {</td>
</tr>
<tr>
<td>double my_balance</td>
<td>my_balance = amt;</td>
</tr>
<tr>
<td></td>
<td>my_name = name;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
**Example: Bank Account**

```cpp
struct bankAccount {
    double my_balance = 0.0;
    string my_name = "?NAME?";

    bankAccount() {
    }

    bankAccount(string name, double amt) {
        my_balance = amt;
        my_name = name;
    }

    void setName(string name) {
        my_name = name;
    }

    void withdraw(double amt) {
        my_balance -= amt;
    }

    void deposit(double amt) {
        my_balance += amt;
    }

    double balance() {
        return my_balance;
    }

    string name() {
        return my_name;
    }

    bankAccount Howie, John("john", 500.0);
```
### Example: Bank Account

<table>
<thead>
<tr>
<th>bankAccount</th>
<th>bankAccount::bankAccount()</th>
</tr>
</thead>
<tbody>
<tr>
<td>void withdraw( double )</td>
<td>my_balance = 0.0;</td>
</tr>
<tr>
<td>void deposit( double )</td>
<td>my_name = &quot;?NAME?&quot;;</td>
</tr>
<tr>
<td>double balance( )</td>
<td>}</td>
</tr>
<tr>
<td>string name( )</td>
<td>bankAccount::bankAccount(</td>
</tr>
<tr>
<td>void setName( string )</td>
<td>string name,</td>
</tr>
<tr>
<td>string     my_name</td>
<td>double amt ) {</td>
</tr>
<tr>
<td>double     my_balance</td>
<td>my_balance = amt;</td>
</tr>
<tr>
<td></td>
<td>my_name = name;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

```c++
bankAccount Howie, John( "john", 500.0 );
```
Time For Our Next Demo!

• Radio.cpp

(See Handout For Example 2)
Summarizing Our Second Demo!

• Constructors Are Typically Overloaded Functions
• Generally, Constructors Call One Another
• Generally, A Default Constructor Should Always Be Defined
• Parameter-Less Class Definition Should Not Use Empty ( )
  – looks like a function invocation that cannot be found
Enumerations

- An Enumerated Type Is A Set Of `int` Constants That Make Up The Legal Values For A Type
- These Types Can Be Used Whereever Constants Are Allowed
  - `switch` Statement
Enumerations

• Examples:
  
  ```
  enum Day {
    SUN=1, MON=2, TUE=3, WED=4, THU=5,
    FRI=6, SAT=7
  };
  enum Color {
    RED, BLACK, BLUE, YELLOW
  };
  ```
Time For Our Next Demo!

• Enumeration.cpp

(See Handout For Example 3)
Summarizing Our Third Demo!

• Enumerations Define A List Of Constants
• The Same Constant Value May Be Defined Many Times In One Enumeration
Default Parameter Values

• Any Function Can Supply Default Parameter Values
• Generally, Default Values Prevent Having To Write Overloaded Function Definitions
• All Default Arguments Must Be Grouped At The End Of A Function Definition
Default Values

• Examples:
  ```c
  void func( double d=0.0 );
  void calc( int arg1,
              int arg2,
              int arg3=0,
              int arg4=1 );
  ```

• Valid Invocations
  ```c
  func();         func( 12.5 );
  calc( 12, 1 );
  ```
Time For Our Next Demo!

• DefaultValues.cpp

(See Handout For Example 4)
Summarizing Our Fourth Demo!

• Default Arguments Can Be Mixed With Normal Parameters

• All Default Arguments Must Be Grouped At The End Of A Function Definition

• Ain’t No Way To Specify A Default Argument Value Without Specifying All The Default Values That Are Listed BEFORE That Argument
Summary

• Defining Classes
• Members, Methods And Scope Resolution
• Delivering Classes
• Constructors
• Enumerations
• Default Parameter Values