

# NET2006: Object-Oriented Programming

## Lab 4: Nesting Classes

### Overview

This lab work will be completed in pairs. The task will be based on three questions in your textbook: page 260: #4, #5 and #6. The three questions are related, and ultimately, you will submit a single program with a single set of documentation that blends the three. The *main()* routine in this lab will be large enough to thoroughly exercise the *test plan* that you create. Here are the essential elements:

- Create a **Date** class.
- Create an **Employee** class that uses the **Date** class.
- Create a **main()** routine that exercises these classes.

The supporting documentation will include:

- *Program Statement (your assumptions)*
- Memory Map
- *Test Plan*
- Only one function will require an *Algorithm (PDL)*: the routine responsible for validating a date.

### Details: Date class

The **Date** class will have the following characteristics:

- Three data members to track the year, month and day of the month.
- A member function will input values from the keyboard (with suitable prompts to the user).
- A member function will allow a programmer to set the date by sending three arguments to the object (the three arguments are the year, month and day).<sup>1</sup>
- Whenever the date values are to be captured (from either the keyboard or through arguments to a **Set()** function call, your class must validate the values being entered. For example, the date: 2008-15-38 is clearly in error. Your class should reject an attempt to set such values<sup>2</sup>. Since validation must be done by both the keyboard-oriented capture and the programmer-oriented capture, it makes sense to put the validation routine in its own separate member function and call it from those two input routines.

Build the **Date** class first, even before starting the **Employee** class. You can test the **Date** class with a simple implementation of **main()**. Your test plan will have to account for a number of cases: a reasonable value for year, a month from 1 to 12, day of the month one of: 28, 29, 30 or 31 (and the correct value for day of the month for February will depend on the year – to account for leap years).

<sup>1</sup> Your textbook sometimes uses poor, non-standard names for functions. For example, in the **Distance** class, the author uses the function name **getdist()** for input. In today's world of object-oriented programming, the function name **Get()** is used to retrieve data from an existing object. On the other hand, **Set()** is used to change the values in variables stored in the object.

<sup>2</sup> Checking the value for month is simple. It can only be a value from 1 to 12. Checking the day-of-the-month is trickier. For some months, 31 is acceptable; for all months 28 is acceptable. The value 29 is acceptable for all months except February (unless it's a leap year, in which case it's acceptable for February as well). There are many ways to implement this error-checking. I'll leave it to you to decide.

### Details: Employee class

After finishing and testing the **Date** class you can start the **Employee** class. (You have to finish and test the **Date** class first since you intend using the **Date** class in the **Employee** class.) The **Employee** class will have the following characteristics:

- Four data members to track the employee number, the employee's salary, the employee's job category (**manager**, **labourer**, **secretary**, etc.) and the employee's date of hire<sup>3</sup>.
- A member function will input values from the keyboard (with suitable prompts to the user).
- A member function will allow a programmer to set the values in an employee object.
- Decide on a format for employee number. Imagine that you work in the *Personnel Department* of a company and are building a program to manage employee data. Your company as a particular format / standard for employee numbers<sup>4</sup>. During the input / set processes, validate the employee number. Of course, you need to identify the format of the employee number in your statement of assumptions. For example, is there to be a limit to the number of digits?
- A member function will display the contents of an **Employee** object.

### Details: main() Routine

The function **main()** is used as a tool to test your classes thoroughly. On the one hand, you should create **main()** to be as small and concise as possible. On the other hand, **main()** must be comprehensive enough to test all categories in your test plan. Create at least two **Employee** objects. One will have its data members established by asking the user to enter values through the keyboard. The other object will have its data members established by sending predefined arguments to the **Set()** function.

### Details: Documentation

#### Problem Statement (assumptions)

You will likely have several assumptions:

- How will your date validation function handle errors? You have two choices: the validation function can correct the error by prompting the user to enter corrections. Alternatively, the validation function can send back a **bool** return value: **true** for success; **false** for error. The code that called the validation routine can then act on that return value.
- What is the format of an Employee ID number?
- What is the purpose of **main()**?

<sup>3</sup> The first two items are basic types: **int** and **float** (or **double**). The third will be of type: **Date** (assuming that you named your date class that way).

<sup>4</sup> The format might require a 4 digit employee number.

## Algorithm

For the one **Date** validation function, you must identify the following:

- **Precondition:** What must exist in memory before the function is called (that is variables and objects)? What do these variables and objects represent?
- **Variables Sent as Arguments in Call to Function:** Is anything sent as an argument? If so, what is the data type of each argument? What is its purpose? Is it sent using *pass-by-value* or *pass-by-reference* techniques?
- **Actions of Function:** What work does the function do?
- **Postcondition:** What has changed in memory as a result of the function's actions?
- **Variables Returned by Function:** Is anything being returned? If so, what is being returned through the return statement.(it's data type and purpose)? If variables were *passed-by-reference*, was it for the purpose of modifying the original variable.

The items shown above will become part of the function header in your C++ code.

## Test Plan

Look to the class details, details of **main()**, and the problem statement (assumptions). With the **Date** validation routine, you should list suitable values to check the correctness.

## Memory Maps

The memory maps must show the **Employee** objects that were created in **main()**.

## Submission Standards

- Due date is posted on my website.
- Cover page:
  - Course Name / Number (including Lab Section)
  - Lab Number
  - You and your partner's names
  - Date of Submission
- Problem Statement (Assumptions)
- Test Plans
- Algorithm for **Date** Validation Function only
- Memory Maps using Visio
- C++ source code

## Submission Process

- Staple all paper components together with the evaluation sheet.

## Evaluation Criteria

See the attached marking guide.

# NET2006: Object-Oriented Programming

## Lab 4: Working With Classes

Your lab assignment has been evaluated using the following criteria: This lab is to be completed with your partner. You will receive the same mark (unless there is evidence that one of the partners did not make a real contribution). Look to Lab 1 for coding standards (sample code, variable naming conventions, etc.)

- Problem Statement (Assumptions)
- Test Plan
- Algorithm
- Memory Map (Use **Visio** for bonus marks)
- Program, class and function headers (remember, *main()* is a function), including history.
- Comment each significant line of code. Comment each related section of your program.
- Variable naming as defined in handout.
- Indentation follows standard in handout.
- (2 marks) Program performs runs correctly from source code (Including meaningful user prompts).

**Subtotal /10**

- Penalties (up to 3 marks deducted): Penalty for redundant code, unused code, unnecessary extra steps. Penalty for poor organization and binding of program and documentation. (Individual Adjustment: Minimal or no participation. This will adjust the grade of an individual within the group. The deduction could up to the full marks available for the lab.)

- Bonus marks (up to 3 marks added): Clever, clear, useful enhancements. (Individual Adjustment: Clear evidence of supportively explaining C++ concepts to partner.) Use of **Visio** for memory map.

**Total /10**