# PEN203

Classes: A Deeper Look,

**C How to Program**
**Deitel & Deitel**

## Outline

## Introduction

- There are three types of handles on an object
  - Name of an object
  - Reference to an object
  - Pointer to an object
- Class functions
  - Predicate functions
  - Utility functions

## Time Class Case Study

- Preprocessor wrappers are used to prevent class definition from being included more than once.
  - #ifndef
    - Skip the code if it has been included already.
  - #define
    - Define a name so the code will not be included again
  - #endif
- Multiple definition errors are eliminated.

## Time Class Case Study

```
// Fig. 20.1: Time.h
// Declaration of class Time.
// Member functions are defined in Time.cpp

// prevent multiple inclusions of header file
#ifndef TIME_H
#define TIME_H

// Time class definition
class Time
{
public:
   Time(); // constructor
   void setTime( int, int, int ); // set hour, minute and second
   void printUniversal(); // print time in universal-time format
   void printStandard(); // print time in standard-time format
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

## Time Class Case Study

```
// Fig. 20.2: Time.cpp
// Member-function definitions for class Time.
#include <iostream>
using std::cout;

#include <iomanip>
using std::setfill;
using std::setw;

#include "Time.h" // include definition of class Time from Time.h

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time()
{
   hour = minute = second = 0;
} // end Time constructor

// set new Time value using universal time; ensure that
// the data remains consistent by setting invalid values to zero
void Time::setTime( int h, int m, int s )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
} // end function setTime
```

## Time Class Case Study

```

// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
      << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
      << second << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

## Time Class Case Study

```
// Fig. 20.3: fig20_03.cpp
// Program to test class Time.
// NOTE: This file must be compiled with Time.cpp.
#include <iostream>
using std::cout;
using std::endl;

#include "Time.h" // include definition of class Time from Time.h

int main()
{
   Time t; // instantiate object t of class Time

   // output Time object t's initial values
   cout << "The initial universal time is ";
   t.printUniversal(); // 00:00:00
   cout << "\nThe initial standard time is ";
   t.printStandard(); // 12:00:00 AM

   t.setTime( 13, 27, 6 ); // change time

   // output Time object t's new values
   cout << "\n\nUniversal time after setTime is ";
   t.printUniversal(); // 13:27:06
   cout << "\nStandard time after setTime is ";
   t.printStandard(); // 1:27:06 PM

   t.setTime( 99, 99, 99 ); // attempt invalid settings
```

## Time Class Case Study

```cpp

   // output t's values after specifying invalid values
   cout << "\n\nAfter attempting invalid settings:"
      << "\nUniversal time: ";
   t.printUniversal(); // 00:00:00
   cout << "\nStandard time: ";
   t.printStandard(); // 12:00:00 AM
   cout << endl;
   return 0;
} // end main
```

```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

## Time Class Case Study

- Member functions are declared in a class definition but defined outside of class definition
  - Still in the class scope
  - Can be accessible by other member functions of the class directly
  - Outside functions can access member functions using:
    - Object of the class
    - Reference to an object of the class
    - Pointer to an object of the class
    - Binary scope resolution operator

## Time Class Case Study

- Using class Time
  - Time time1;
  - Time timeAr[20];
  - Time &timeRef = time1;
  - Time *timePtr = &time1;

## Class Scope and Accessing Class Members

- Class scope contains:
  - Data members
  - Member functions
- Nonmember functions are defined at file scope
- Variables declared in a member function
  - Have block scope
  - Known only in that function

## Class Scope and Accessing Class Members

- Dot member selection operator (.)
  - Accesses the object’s member
  - Used with an object’s name or with a reference to an object
- Arrow member selection operator (->)
  - Accesses the object’s members
  - Used with a pointer to an object

## Access Functions and Utility Functions

- Access functions can read or display data
- They can test the truth or falsity of conditions
  - They are also called predicate functions
- Utility functions are private member functions that help the operation of public member functions
- Utility functions are not part of the class public interface

## Constructors with Default Arguments

- To initialize data members to a consistent state, constructors can specify default arguments
- If no values are provided in a constructor call, default arguments are used.
- They can be invoked with no arguments
- Max one default constructor per class

## Destructor

- A special member function like constructor
- Name is the tilde character (~) followed by the class name
- Destructors are called when an object is destroyed
- They receive no parameter and returns no value
- A class may have only one destructor
- If a destructor is not provided, the compiler creates one empty destructor.

## When Constructors and Destructors are Called

- Constrcutors and destructors are called implicitly by the compiler
- Generally destructor calls are made in the reverse order of the corresponding constructor calls
- Storage classes of objects can alter the order.

## Default Memberwise Assignment

- Assignment operator (=) can be used to assign an object of class to another object of the same type.
- Each data member of the right object is assigned to the same data member of the left object.
- Can crate problem if data members contain pointers to dynamically allocated memory