Ch 13: Introduction to Classes
Part 2

CS 2308
Fall 2011

Jill Seaman

Lecture 11

Procedural Programming

- Data is stored in variables
  - Perhaps using arrays and structs.
- Program is a collection of functions that perform operations over the variables
  - Good example: book inventory program
- Usually variables are passed to the functions as arguments
- Focus is on organizing and implementing the functions.

Procedural Programming: Problem

- It is not uncommon for
  - program specs to change
  - representations of data to change for internal improvements.
- As procedural programs become larger, more complex, it is difficult to make changes.
  - A change to a given variable or data structure requires changes to all of the functions operating over that variable or data structure.

Object Oriented Programming: Solution

- An object contains
  - data
  - functions that operate over the data
- Code outside the object can access the data only via the member functions.
- If the representation of the data in the object needs to change:
  - The member functions must be redefined to handle the changes.
  - The code outside the object does not need to change, it accesses the object in the same way.

Object Oriented Programming: Concepts

- **Encapsulation**: combining data and code into a single object.
- **Data hiding** (or **Information hiding**) is the ability to hide the details of data representation from the code outside of the object.
- **Interface**: the mechanism that code outside the object uses to interact with the object.
  - The member functions
  - Specifically outside code needs to know only the function prototypes.

Object Oriented Programming: Real World Example

- In order to drive a car, you need to understand only its interface:
  - ignition switch
  - gas pedal, brake pedal
  - steering wheel
  - gear shifter
- You don't need to understand how the steering works internally.
- You can operate any car with the same interface.

Classes and Objects

- A class is like a blueprint for an object.
  - a detailed description of an object.
  - used to make many objects.
  - these are called **instances** of the class.
- For example, the String class in C++.
  - Make an instance or two:

```
String cityName1("Austin"), cityName2("Dallas");
```

  - use member functions to manipulate the objects:

```
int size = cityName1.length();
cityName2.insert(0,"Big ");
```


Example class declaration

- Example: time.h, modified

```
// file time.h
#include <string>
using namespace std;

  class Time          //new data type
  {
    private:
      int hour;
      int minute;
      void addHour();

    public:
      void setHour(int);
      void setMinute(int);
      int getHour();
      int getMinute();

      string display();
      void addMinute();
  };
```

Access rules

- Private members can be accessed only from other member functions within the class
  - member variables (attributes) are declared private, to hide their definitions from outside the class.
  - certain functions are declared public and provide (limited) access to the hidden/private data.
- The public members provide the interface which defines how code outside the class can use instances (objects) of the Time data type.

Defining Member Functions

- the Time.cpp file:

```
// file time.cpp
#include <sstream>
#include <iomanip>
using namespace std;

#include "time.h"

void Time::setHour(int hr) {
  hour = hr;            // hour is a member var
}
void Time::setMinute(int min) {
  minute = min;         // minute is a member var
}

int Time::getHour() {
   return hour;
}
int Time::getMinute() {
   return minute;
}
```

Defining member functions

- Member function definitions occur OUTSIDE of the class definition, usually in a separate file.
- The name of each function is preceded by the class name and :: operator
  - Time::setHour(int hr)
- Accessor functions (a.k.a. “getters”)
  - returns a value from the object (without changing it)
- Mutator functions (a.k.a. “setters”)
  - Changes values of member variables.

Defining an instance of the class

- ClassName objectname:

```
Time t1;
```

- This defines t1 to contain an object of type Time (the values of hour and minute are not set).
- Access public members of class with dot notation:

```
t1.setHour(3);
t1.setMinute(41);
t1.addMinute();
```

- Use dot notation OUTSIDE class only.

Setters and getters: what's the point?

- Why have setters and getters that just do assignment and return values?
- Why not just make the member variables public?

- Setter functions can validate the incoming data.
  - setMinute can make sure minutes are between 0 and 59 (if not, it can throw an *exception*).
- Getter functions could act as a gatekeeper to the data (validate "user") or provide type conversion.

Separating Specs from Implementation

- Class declarations are usually stored in their own header files (time.h) Name is usually same as class.
- Member function definitions are stored in a separate file (time.cpp) called the class implementation file (must #include the header file).
- Any program/file using the class should #include the class's header file.

Separating Specs from Implementation

- To share/re-use a class, you can give someone the header file and the class implementation file.
  - You don't have to give them your whole program, or try to cut out relevant parts.
- Other programmer can just #include the header (and compile all the files together).
- If the class implementation has to be changed, only the class implementation file needs to change. Then it can be recompiled/linked to the other files.