CS250 Intro to CS II

Spring 2013

Chapter 13
Introduction to Classes

- Reading: pp. 705-723, 728-734
- Good Problems to Work: p. 727 13.1, 13.2, 13.3, 13.4, 13.5,

Introduction to Classes

- Procedural programming is a programming methodology centered on procedures (or actions) taking place in a program

- Object-oriented programming is a programming methodology centered on objects created from user-defined data types that encapsulate data and functions together

OOP

The class is a C++ construct used to create objects which are fundamental to object-oriented programming (OOP).

- OOP hides the details of objects from objects of other types
- When an object needs information from another object or needs another object to perform a task, a message is sent as a form of communication between objects
- As a result, object-oriented programs can be written more generically than structured programs
- Usually, making changes to the object-oriented programs is easier than changing structured programs

Class Definition

class defintion

A class is a user-defined datatype that is defined by the programmer. A class consists of variables and functions with a general format as follows:

```
class ClassName
{
  Declarations for member variables and member
  functions
};
```

Person Class

```
class Person
{
  public:
   int mAge;                    // member variable

   int getAge ();               // member function prototype
   int getBirthYear ();         // member function prototype
};

int main()
{
  Person cPerson;               // creates a Person object
  cPerson.mAge = 28;
  cout << "person is: " << cPerson.getAge();
  cout << "person was born in: " << cPerson.getBirthYear();
}
```

Person Class Definitions

```
int Person::getAge ()
{
  return mAge;
}

int Person::getBirthYear ()
{
  return 2013 - mAge;
}
```

public: versus private:

- Class data members and member functions can be either private or public
- Private data members and member functions can only be accessed within the class in which they are defined
- Public data members and member functions can be accessed from inside or outside of the class in which they are defined

Revised Person Class

```
class Person
{
  public:
    void setAge (int);          // member function prototype
    int getAge ();              // member function prototype
    int getBirthYear ();        // member function prototype

  private:
    int mAge;                   // private member variable
};

int main()
{
  Person cPerson;               // creates a Person object
  cPerson.mAge = 28;            // error
  cPerson.setAge (28);          // correct setting of age
```

Mutator

- A mutator is any method that can change the value of a member variable

```
void Person::setAge (int age)
{
  mAge = age;
}
```

Accessor

- An accessor is a method that uses a class member but does not change the member value

```
int Person::getAge ()
{
  return mAge;
}
```

Revised Person Class

- Using the reserved word const after a function prototype prohibits the function from changing any data directly in the class

```
class Person
{
  public:
    void setAge (int);          // mutator
    int getAge () const;        // accessor
    int getBirthYear () const; // accessor

  private:
    int mAge;                   // private member variable
};
```

What is a Rectangle?

- A rectangle has a width and length
- Operations we might want to perform on a rectangle include
  a. setting the length or width to a value
  b. getting the length or width
  c. calculating the area
  d. calculating the perimeter

Representation in a Procedural Language

- Basically length and width are set/reset somewhere and getArea and getPerimeter can be called when needed
- length and width are separate from the functions

```
double length, width;

double getArea (double, double);
double getPerimeter (double, double);
```

OO Representation

```
class Rectangle
{
  public:
    void setLength (double);
    void setWidth (double);
    double getLength () const;
    double getWidth () const;
    double getArea () const;
    double getPerimeter() const;

  private:
    double mLength;
    double mWidth;
};
```

class Rectangle Questions

Q1: How many members does class Rectangle have? List them.

Q2: How many methods does class Rectangle have? List them.

Q3: How many mutators does class Rectangle have? List them.

Q4: How many accessors does class Rectangle have? List them.

class Rectangle Questions

Q5: Where can we define any member functions?

Q6: Show how getArea is defined both ways.

class Rectangle Questions

Q7: How do we create objects of class Rectangle?

a) A regular object

b) An array of 50 objects

class Rectangle Questions

Q8: Write the C++ code that shows how to use each of the objects created in Q7 a) through b).

Constructors

- Special member function to initialize data members
- Has the same name as the class
- Does not have a return value
- The constructor is called whenever an object of that class is created (instantiated)

Constructor for Rectangle

What might the constructor for class Rectangle look like?

1) Add **`Rectangle ();`** as a public member prototype of Rectangle
2) Add implementation code as:

```
Rectangle::Rectangle ()
{
  mLength = mWidth = 0;
}
```

OO Features

- Information hiding
  - Separate the implementation (.cpp) from the interface (.h)
  - Objects are concerned with the interface, for example what functions are available to manipulate the data
  - Objects are not concerned with the implementation. They do not care how the functions do what they do

Overloaded Constructors

- Constructors and functions can be overloaded (multiple definitions)
- We could have multiple constructors in the Rectangle class, each of which accepts a different number of arguments
- The appropriate constructor will be chosen based on the number of arguments used when creating the object

Overloaded Rectangle Constructor

```
// default constructor
Rectangle::Rectangle ()
{
  mLength = mWidth = 0;
}

// overloaded constructor
Rectangle::Rectangle (double length, double width)
{
  mLength = length;
  mWidth = width;
}
```

Default Constructor

- The default constructor is the constructor with no arguments
- If you do not create any constructors in your class, then the default constructor will be created for you
- If you only have a constructor that takes arguments, then there is no default constructor
- It is good programming practice to always create a default constructor, why?

Default Arguments

- You can set default arguments to constructors
- In the class definition, the constructor prototype will be
  - `Rectangle (double = 0.0, double = 0.0);`
- The function definition will be

```
Rectangle::Rectangle (double length,
                      double width)
{
   mLength = length;
   mWidth = width;
}
```

Using Default Arguments

- By having default arguments in the constructor, we can now create objects of the Rectangle class as follows:

```
Rectangle cR1;

Rectangle cR2(10.0);

Rectangle cR3(5.0, 25.0);
```

Rectangle Interface

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle
{
  public:
    Rectangle (double = 0.0, double = 0.0);
    void setLength (double);
    void setWidth (double);
    double getLength () const;
    double getWidth () const;
    double getArea () const;
    double getPerimeter() const;

  private:
    double mLength;
    double mWidth;
};

#endif
```

Rectangle Implementation

```
#include "Rectangle.h"

Rectangle::Rectangle (double length, double width)
{
  mLength = length;
  mWidth = width;
}

void Rectangle::setLength (double length)
{
  mLength = length;
}
```

Problem

- Grab the files Rectangle.h, Rectangle.cpp, and main.cpp from the folder Rectangle found in the Public directory
- Add a project Rectangle to your CS250InClass solution
- Place the three files appropriately into the project, build, and run