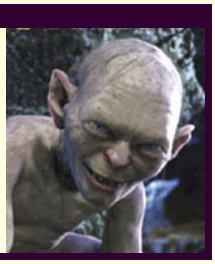
Lecture 14: Classes

Todd Wittman
PIC 10A



Sec 5.1: Classes

- Recall that a class is a suped-up data type equipped with special member functions.
- We call an instance of a class an object.
- The string class is an example of a class.

```
string my_string = "hobbit";
```

class name object name value of object

- The insert function is an example of a member function.

```
my_string.insert(3,"orc");     //Use with a period after object name.
```

- In Chapter 3, we saw some classes defined by your textbook author.
  - The Time class
  - The Employee class
  - Graphics classes: Point, Circle, Line, Message

Why Use Classes?

- In Chapter 6, we'll learn how to create our own classes.
- Why bother? OK, that's a fair question.
- Suppose you're a computer graphics programmer.
- What other graphics objects would you like to have available besides Line, Circle, Point?
  - Triangle? Rectangle? Curve? Gollum?
- What member functions would you like to add to the graphics objects?
  - changeColor? fillObject? setFontSize?
- For business and banking, it might be useful to have a Database object to store numbers and make calculations. You can think of an Xcel spreadsheet as one of Microsoft's objects.

Sec 5.2: Interfaces

- Declaring a class is a lot like declaring a function.

```
class Class_name {
    public:
        constructor declarations
        member function declarations
    private:
        data fields
};
```

Functions that create the object.

Class functions.

Internal variables and functions.

Note the weird semi-colon at the end. Only for class declarations.

- Public fields are accessible outside the class.
  - e.g. my_point.get_x ( )
- Private fields are only accessible to the class itself, much like local variables.
  - e.g. the actual variable x in my_point

The Point Class

Recall the Point class has the following functions available.

Constructor function	Point (double x , double y)	Constructs a point at location (x,y).
Accessor functions	double p.get_x ( )	Returns the x-coordinate of p.
	double p.get_y ( )	Returns the y-coordinate of p.
Mutator function	p.move (double dx , double dy)	Moves the point p by (dx , dy).

- Constructors create the object.
- Accessors get the data stored in the object.
- Mutators change the data stored in the object.

The Point Class

- The declaration for the Point class looks like this.

```
class Point  {
    public:
        Point (double xval, double yval);
        double get_x() const;
        double get_y() const;
        void move(double dx, double dy);
    private:
        double x;
        double y;
};
```

Constructor function

Accessor functions

Mutator function

Private (local) variables

Weird semi-colon. Don't forget that.

- We can't use x and y directly. To access the values, we use the accessor functions get_x and get_y. To change the values of x and y, we use the mutator function move.
- This prevents the rest of the program from accidentally changing the value of x and y.

Sec 5.3: Encapsulation

- Placing variables and functions in the private section of a class is called encapsulation.
- Then we can use the variable names x and y throughout the program freely.

```
int ccc_win_main ( )  {
      Point my_point = Point (1,3);
      int x = 22;
```

Calls constructor.

Won't affect the x value of any of our points.

- The value of my_point's x is still x=1.
- To get that value, we use the accessor.

```
      int my_x = my_point.get_x( );
```

Calls accessor.

- To change the value of my_point's x to 22, use the mutator:

```
      my_point.move(21,0);
```

Calls mutator.

The const Modifier

- Note the accessor functions have the word const after their declaration.

```
class Point  {
   public:
      Point (double xval, double yval);
      double get_x() const;
      double get_y() const;
      void move(double dx, double dy);
   private:
      double x;
      double y;
};
```

Declares return type x as constant.

Declares return type y as constant.

- This is another way to prevent the program from accidentally changing the value of x and y.
- Always add the word const to the end of your accessors.

The Product Class

- Suppose we're cataloging and comparing different products for *Consumer Reports*.
- It would be helpful to store the various items for sale in some standard class.
- What data should the product store?
  - Product name (string)
  - Product price (double)
  - Product score on 0-10 scale (int)
- What functions would we like to have?
  - Create new product (constructor).
  - Read in a new product's details (mutator).
  - Compare two products (accessor).
  - Print out a product's details (accessor).
- Which of our functions should get the const modifier?

The Product Class Declaration

```
class Product  {
   public:
        Product();
        void read();
        bool is_better_than(Product b) const;
        void print() const;
   private:
        string name;
        double price;
        int score;
};
```

- Place in your program above main(), like a function.

Sec 5.4: Member Functions

- We've declared the class, but we haven't defined any of its functions yet.
- The general form of a member function definition is:

```
return_type ClassName::function_name (parameters) const {
    ** STATEMENTS **
}
```

Optional.
Add if it's an accessor.

- We add the const modifier at the end if it's an accessor. Should match the consts used in the declaration.
- By adding ClassName:: before the function name, we make sure that the program knows this is a member function for that class.

Product's read function

- Read in the name, price, and score of a product.

```
void Product::read( )  {
        cout << "Please enter the model name: ";
        getline(cin, name);
        cout << "Please enter the price: ";
        cin >> price;
        cout << "Please enter the score: ";
        cin >> score;
        string remainder;
        getline(cin, remainder);
}
```

- Note we do not declare the variables name, price, and score. They're already declared by the class declaration.
- What does the variable remainder do?
- Why do we use getline once and cin twice.
- What would the other functions look like?