
Lecture 2: Introducing Classes

Dragan Mirkovic
Department of Computer Science
University of Houston

D. Mirkovic, C++ Programming, Spring 2005

Announcements

- Today:
 - Fundamental components of object-oriented design
 - Ch 2. In Irvine.
 - Introducing Classes
 - The slides are based on the Kip Irvine lecture slides.

D. Mirkovic, C++ Programming, Spring 2005

Object-Oriented Programming

- Literally, "programming with objects"
- Modeling the real world
 - objects in programs are patterned after objects in project specifications
- Originally, C++ was called "C with classes"
 - See: The Design and Evolution of C++ by Stroustrup.

D. Mirkovic, C++ Programming, Spring 2005

Classes and Objects

- A class is a user-defined data type.
 - It provides a "template" or "design" for multiple objects of the same type.
- An object is an instance of a class.
 - each object has a unique storage location, and unique values for its attributes
- A class contains attributes and operations:
 - (continued...)

D. Mirkovic, C++ Programming, Spring 2005

Attributes and Operations

- Attributes are characteristics shared by all instances of a class
 - implemented in C++ as variables (also called data members)
- Operations are implemented in C++ as functions (also called member functions)
- (Real-world objects have attributes and operations)

D. Mirkovic, C++ Programming, Spring 2005

Point Class Example

- A class contains attributes (variables) and operations (functions):

```
class Point {  
    void Draw();  
    void MoveTo( int x, int y );  
    void LineTo( int x, int y );  
  
    int m_X; ← attributes  
    int m_Y;  
}; ← don't forget the semicolon!
```

operations

D. Mirkovic, C++ Programming, Spring 2005

Encapsulation

- Encapsulation is the act of hiding a class's implementation details from users of the class.
- Class users should only interact with a class via its interface.
- The implementation might change during its lifetime, but this should have only a minimal effect on class users.
- The private specifier enforces encapsulation in C++ classes.

D. Mirkovic, C++ Programming, Spring 2005

Public and Private Access Specifiers

- All members following a public specifier are visible outside the class
 - for example, a public member is visible from `main()`, as in the case of `cin.get()`:
 - `cin.get();`
- All members following a private specifier are hidden from functions outside the class.
 - they may only be referenced by functions inside the current class
- (There is another access specifier -- protected, covered in Chapter 14.)

D. Mirkovic, C++ Programming, Spring 2005

Point Class, revised

- This version uses public and private:

```
class Point {
public:
    void Draw();
    void MoveTo( int x, int y );
    void LineTo( int x, int y );

private:
    int m_X;
    int m_Y;
};
```

D. Mirkovic, C++ Programming, Spring 2005

Automobile Class Example

- Imagine that you want to program an automobile simulation:
- Attributes: make, number of doors, number of cylinders, engine size
- Operations: input, display, start, stop, check_gas
- The choice of which attribute and operations to include depends on the needs of the application

D. Mirkovic, C++ Programming, Spring 2005

Automobile Class

```
class Automobile {
public:
    Automobile();
    void Input();
    void set_NumDoors( int doors );

    void Display();
    int get_NumDoors();

private:
    string Make;
    int    NumDoors;
    int    NumCylinders;
    int    EngineSize;
};
```

D. Mirkovic, C++ Programming, Spring 2005

Types of Member Functions

- An accessor is a function that returns a value from its object, but does not change the object
- A mutator is a function that modifies its object
- A constructor is a function having the same name as the class
 - It executes when an instance of the class is created

→
code example...

D. Mirkovic, C++ Programming, Spring 2005

Automobile Class

```
class Automobile {  
public:                                // public functions  
    Automobile();                      // constructor  
    void Input();                      // mutator  
    void set_NumDoors( int doors );    // mutator  
    void Display();                   // accessor  
    int get_NumDoors();               // accessor  
  
private:                               // private data  
    string Make;  
    int    NumDoors;  
    int    NumCylinders;  
    int    EngineSize;  
};
```

D. Mirkovic, C++ Programming, Spring 2005

Creating an Object

- When you declare a variable using a class as its data type, you are creating an instance of the class
- An instance of a class is also called an object or a class object
- In the next example, we create an Automobile object and call its member functions

—————→

code example...

D. Mirkovic, C++ Programming, Spring 2005

Creating and Accessing an Object

```
void main()
{
    Automobile myCar;

    myCar.set_NumDoors( 4 );

    cout << "Enter all data for an automobile: ";
    myCar.Input();

    cout << "This is what you entered: ";
    myCar.Display();

    cout << "This car has "
         << myCar.get_NumDoors()
         << " doors.\n";
}
```

D. Mirkovic, C++ Programming, Spring 2005

Constructors

- A constructor is a function having the same name as its owning class
- A constructor executes when an object is created
- A default constructor has no parameters
- A constructor usually initializes the class data members
- If an array of objects is created, the default constructor is called for each object:

```
Point drawing[50];
// calls default constructor 50 times
```

D. Mirkovic, C++ Programming, Spring 2005

Constructor Implementation

- A default constructor for the Point class could initialize X and Y:

```
class Point {  
public:  
    Point() {  
        m_X = 0;  
        m_Y = 0;  
    }  
private:  
    int m_X;  
    int m_Y;  
};
```

D. Mirkovic, C++ Programming, Spring 2005

Out-of-Line Functions

- All member functions must be declared (prototyped) inside the class definition block
- Implementations of non-trivial functions are usually outside the class definition, in a separate CPP file.
- Point constructor example:

```
Point::Point()  
{  
    m_X = 0;  
    m_Y = 0;  
}
```

D. Mirkovic, C++ Programming, Spring 2005

Automobile Class (review)

```
class Automobile {
public:
    Automobile();
    void Input();
    void set_NumDoors( int doors );

    void Display() const;
    int get_NumDoors() const;

private:
    string Make;
    int    NumDoors;
    int    NumCylinders;
    int    EngineSize;
};
```

D. Mirkovic, C++ Programming, Spring 2005

Automobile Function Implementations

```
Automobile::Automobile()
{
    NumDoors = 0;
    NumCylinders = 0;
    EngineSize = 0;
}

void Automobile::Display() const
{
    cout << "Make: " << Make
         << ", Doors: " << NumDoors
         << ", Cyl: " << NumCylinders
         << ", Engine: " << EngineSize
         << endl;
}
```

D. Mirkovic, C++ Programming, Spring 2005

Input Function Implementation

```
void Automobile::Input()
{
    cout << "Enter the make: ";
    cin >> Make;
    cout << "How many doors? ";
    cin >> NumDoors;
    cout << "How many cylinders? ";
    cin >> NumCylinders;
    cout << "What size engine? ";
    cin >> EngineSize;
}
```

D. Mirkovic, C++ Programming, Spring 2005

Overloading the Constructor

- Multiple constructors with different parameter lists:

```
class Automobile {
public:
    Automobile();

    Automobile( string make, int doors,
               int cylinders, int engineSize );

    Automobile( const Automobile & A );
    // copy constructor
```

D. Mirkovic, C++ Programming, Spring 2005

Invoking a Constructor

```
// sample constructor calls:
```

```
Automobile myCar;
```

```
Automobile yourCar("Yugo",4,2,1000);
```

```
Automobile hisCar( yourCar );
```

D. Mirkovic, C++ Programming, Spring 2005

Implementing a Constructor

```
Automobile::Automobile( string p_make, int doors,  
                        int cylinders, int engineSize )  
{  
    Make = p_make;  
    NumDoors = doors;  
    NumCylinders = cylinders;  
    EngineSize = engineSize;  
}
```

D. Mirkovic, C++ Programming, Spring 2005

Constructor with Parameters (2)

- Rarely should parameter names be identical to data member names:

```
NumDoors = NumDoors;           // ??  
NumCylinders = NumCylinders;  // ??
```

- Member variables can be qualified using this, a pointer to the current object:

```
this->NumDoors = NumDoors;  
this->NumCylinders = NumCylinders;
```

D. Mirkovic, C++ Programming, Spring 2005

Initializer Lists

- Use an initializer list to set the values of member variables in a constructor. This is particularly true for member objects:

```
Automobile::Automobile( string make, int doors,  
    int cylinders, int engineSize ) :  
    Make(make),  
    NumDoors(doors),  
    NumCylinders(cylinders),  
    EngineSize(engineSize)  
{  
  
}
```

D. Mirkovic, C++ Programming, Spring 2005

Identifier Naming Scheme

- Microsoft likes to use a standard prefix for their class data members: `m_`
- They also use a prefix letter for the data type:
 - `s` = string
 - `n` = numeric
 - `b` = boolean
 - `p` = pointer
- They sometimes use longer prefixes:
 - `str` = string, `bln` = boolean, `ptr` = pointer, `int` = integer, `lng` = long, `dbl` = double, `sng` = float (single precision)

D. Mirkovic, C++ Programming, Spring 2005

Example: Automobile Data Members

- Here's how the Automobile data members would look with prefixed names:

```
class Automobile {  
  
    private:  
        string m_strMake;  
        int    m_nNumDoors;  
        int    m_nNumCylinders;  
        int    m_nEngineSize;  
};
```

- The advantage is that data members are never mistaken for local variables.

D. Mirkovic, C++ Programming, Spring 2005

Using Const

- Always use the const modifier when declaring a member function if the function does not modify the object's data:

```
void Display() const;
```

- This enables users of your class to use const correctly in their programs
- An object cannot be passed by const reference to a function if inside the function it calls a non-const member function.

—————→
code example...

D. Mirkovic, C++ Programming, Spring 2005

Using Const - Example

```
void ShowAuto( const Automobile & aCar )  
{  
    cout << "Example of an automobile: ";  
    aCar.Display();  
    cout << "-----\n";  
}
```

guarantees not to
modify the parameter
variable

What if Display() is
not a const function?

D. Mirkovic, C++ Programming, Spring 2005

Class Design Ideas

- Descriptive: A class easily matches an entity or process in the problem domain
- Simple: A class describes only the attributes and operations necessary to the problem at hand
- Operations should describe actions or modify class data
- Cohesive: A class is self-contained and does not depend on implementation details in other classes

D. Mirkovic, C++ Programming, Spring 2005

Summary

- Fundamentals of designing and using classes
- Object properties: state, behavior, and identity
- Link and composition relationships
- Object interfaces
- Member functions:
 - Constructors, destructors, operators, selectors, modifiers, and iterators.

D. Mirkovic, C++ Programming, Spring 2005

Homework

- Exercises 1, page 54 and 1-2 on page 57 in text.