An object has three facets: (1) what it is called, (2) what it is, and (3) what it does. If you have ever programmed a computer using a language that allows you to declare a variable to be a particular type, you have, in a sense, used objects. For example, the following statement in C++ declares two integer variables, `first_num` and `second_num`.

```c++
int first_num, second_num;
```

These variables could be thought of as two objects of type int (integer). In fact, the term object variable is commonly used. Relating the three facets of an object to an integer variable, we have: (1) the object is called `int` (at least in C++), (2) the object is the integer number, and (3) the object does input, output, arithmetic, and comparisons. Note that “what the object does” would be the available operations to manipulate integers.

In the strictest sense, a variable of a built-in data type is not an object. Suppose we need a variable of type Rectangle. Of course, there is no language that supports type Rectangle as a built-in data type. We can extend C++ to allow Rectangle variables by writing a description of a Rectangle object. Recall the three facets of an object:

- **What it is called:** Rectangle
- **What it is:** length and width (the data that defines the properties)
- **What it does:**
  - Set (set new values for length and width)
  - Show (display the length and width)
  - Area (compute the area of the rectangle)
  - Perimeter (compute the perimeter of the rectangle)

The four actions that are listed under What a Rectangle can do are known as behaviors or methods. A method is an algorithm or set of operations that specify how an object does something. Basically, it is the code making up a function, usually called a member function in C++. Thus, the Rectangle object has four member functions associated with it.
A program that uses a Rectangle object (Rectangle variable) might need several Rectangle objects or even an array of Rectangle objects, just as you might need one or more integer variables or an array of integers. All Rectangle objects would be similar. They would have the same data, length and width, and the same methods. In order to declare a variable to be type Rectangle, the compiler must know about the Rectangle’s data and methods. In other words, we must specify what a Rectangle class is by giving a class declaration and method definitions.

The class declaration for Rectangle is shown below:

```
lines 12 - 24

class Rectangle // Prototype for a Rectangle object or Class declaration
{
    public:
        void set(int w, int h); // Prototypes for the four member functions
        void show();
        int area();
        int perimeter();
    private:
        int length; // Data members
        int width;
};
```

The class declaration contains descriptions of the members of the class Rectangle. The prototypes for the member functions are given. Also included are the declarations for the data members, length and width. Note that the above class prototype does describe the three aspects of the object: (1) an object of type Rectangle -- what it is called, (2) consisting of length and width -- what it is, (3) performing four functions -- what it does. public and private are access specifiers. A public class member can be directly accessed by any user of the class, that is, any function that declares one or more objects of the class. A private class member can only be accessed by a member function. Note that area() and perimeter() are not inlined in this declaration.

To completely describe the class, we would also need to give the definitions of the four member functions (the methods). The definition of one function followed by a brief explanation is shown below.
In the function's header, the binary scope resolution operator (::) separates the class name and the function name, telling the compiler that class Rectangle owns this set() function. We could have another function set() owned by another class Triangle (Triangle::set()). Note that the rest of the header is as usual, the return type preceding the qualified function name and the formal argument list following the function name. The body of the function consists of two assignment statements that assign the values of the formal arguments, \( l \) and \( w \), to the private data member of an object. Note that this public member function has access to the private data members, length and width.

To reiterate, the class specification consists of the class declaration, including the member function prototypes and data members declarations, and the class method definitions, giving the code for all member functions.

Now that we have described the Rectangle class, we can use it. The user of the class is not a human user but a function, typically the main function. Portions of main() are shown below:

```cpp
int main()
{
    Rectangle pool, garden; // Declare pool and garden to be
    // Rectangle objects
    pool.set(50, 15);       // Send the set message to the pool object
    garden.set(30, 20);    // Send the set message to the garden object
    ...                    // Send the show message to the pool object
    pool.show();           // Send the show message to the pool object
    ...                    // Send the show message to the pool object
}
```

The first line in main() is a type declaration statement. pool and garden are declared to be objects of type Rectangle. Memory is allocated to the data members of each object when the object is declared, not when the class is declared. Think of the class prototype as a template for the
creation of an object. An object is created when the object variable is declared, not when the class is declared.

The next line sends the set message to the pool object. A message is a command sent to an object asking it to do something. Sending a message to an object is basically invoking a member function for that object. Note that the set message is sent to object pool with actual arguments 50 and 15. The same message is sent to object garden with arguments 30 and 20. The show message is sent to object pool with no arguments. The syntax for sending a message to an object is: the object’s name, the dot member operator, followed by the member function name with the actual arguments in parentheses. Note that the member functions can only be invoked through an object. In OOP jargon, action is invoked by sending or passing a message to an object, the receiver, which is responsible for carrying out the action. If the member function takes arguments, the message is accompanied by additional information needed to carry out the action. In response to a message, the receiver, the object, performs a method. Notice that the object is active rather than passive. It takes on responsibilities for itself such as setting itself, printing itself, and computing its area. Please refer to the complete program listing and output starting on page 13-7. Note that very few details are required in main(). Typically, the detailed operations are done by the member functions.
I would visualize the Rectangle class and the two objects as follows:

```
class Rectangle
    Member Functions (Methods)
    set()
    show()
    area()
    perimeter()
```

**Objects: (Instances of Rectangle)**

```
pool
    length
    50
    width
    15

garden
    length
    30
    width
    20
```

The above pictures show the data members of the two Rectangle objects. You can see that there is a `length` belonging to `pool` and a `length` belonging to `garden`. In other words, each Rectangle object has its own `length` and `width`. Recall that memory is allocated to the data members when the object variable is declared not when the class is declared. Note that we could declare many variables of type Rectangle. Also notice that the member functions are represented once for the Rectangle class. With the exception of inline functions, there is one copy of each member function for the class. Each object uses that copy. In other words, there is not a copy of the `set()` function for `pool` and a second copy of the `set()` function for `garden`.

To briefly review what this program did: C++ was extended to handle a Rectangle type consisting of a length and width. The available Rectangle operations are set it, show it, compute its area, and compute its perimeter.
Summary of terminology:

A class is a type of object; a set of objects; a recipe for creating objects; describes the data members and member functions; a programmer-defined type that describes the data and the operations associated with the data; a category of objects defined by a given set of attributes and behaviors. Notice that class members, data and functions, are just ordinary variables and functions that are linked together in a class. An object or instance of the class has its type (name of the class), its data members, and its member functions. Recall the three aspects of an object: what it is called, what it is, and what it does. Note that each object has its own set of data members (state variables or instance variables). However, all objects of one class share one copy of the member functions. A member function is invoked by passing a message to an object.

We have the following three sets of synonyms:

<table>
<thead>
<tr>
<th>object</th>
<th>data member</th>
<th>member function</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance of class</td>
<td>member variable</td>
<td>method</td>
</tr>
<tr>
<td>instantiation</td>
<td>instance variable</td>
<td>behavior</td>
</tr>
<tr>
<td>class object</td>
<td>state variable</td>
<td>message</td>
</tr>
<tr>
<td></td>
<td>attribute</td>
<td></td>
</tr>
</tbody>
</table>

I present this analogy. There is a building contractor who has a blueprint (the class) for a house having three bedrooms, two baths, living room, den, and kitchen (data members). The contractor employs one painter, one carpenter, and one electrician (member functions). The contractor builds ten houses (ten objects of type house are created). Each house (object) has its own kitchen, den, living room, baths, and bedrooms (its own data members). The painter, carpenter, and electrician (the three member functions) are shared by all houses (objects).
// Rectangle: Object-oriented program used to illustrate
// class, object, method, message, and behavior

#include <iostream> // for cin, cout

using namespace std;

// Rectangle class Declaration ///////////////////////////////////////////////////////////////

class Rectangle
{
  public:
    void set(int l, int w); // Prototypes for member functions
    void show();
    int area() { return (length * width); } // inline methods
    int perimeter() { return (2 * length + 2 * width); }
    // Any function defined in a class declaration is automatically inlined.
    public:
    int length; // Declarations of data members
    int width;
};

/* Note that the class declaration allocates no memory for data member,
  length and width.

  A public class member can be directly accessed by any user of the class, that is, any function that declares one or more objects of the class.
  A private class member can only be accessed by a member function.
  Rectangle is an ADT(Abstract Data Type), a programmer-defined type that can be manipulated in a manner similar to a built-in type.
*/

// Member functions definitions ///////////////////////////////////////////////////////////////

void Rectangle::set(int l, int w) // Set length and width of a Rectangle
{
    length = l;
    width = w;
}

void Rectangle::show() // Show length and width of a Rectangle
{
    cout << "Length is " << length << " and width is " << width << endl;
}
// Use Rectangle class

int main()
{
    Rectangle pool, garden; /* Declare pool and garden to be objects of class Rectangle. At this point, memory is allocated for the data members, length and width, of each object. */
    pool.set(50, 15); // Send set message to pool
    garden.set(30, 20); // Send set message to garden
    cout << "Report for pool:" << endl;
    pool.show(); // Send show message to pool
    cout << "Area is " << pool.area() << endl; // Send area message to pool
    cout << "Perimeter is " << pool.perimeter() << endl; // Send perimeter message to pool
    cout << "Report for garden:" << endl;
    garden.show(); // Send show message to garden
    cout << "Area is " << garden.area() << endl; // Send area message to garden
    cout << "Perimeter is " << garden.perimeter() << endl; // Send perimeter message to garden
    return 0;
}

Output:
Report for pool:
Length is 50 and width is 15
Area is 750
Perimeter is 130

Report for garden:
Length is 30 and width is 20
Area is 600
Perimeter is 100

Note:
1. The scope resolution operator (::) is not needed for inline functions.
2. The details are in the methods, not in the user of the class.
3. Organization of an OOP program.
4. Extended C++ to handle Rectangle type.
Stack during `garden.set(30, 20)`:

```
30  l
20  w

set()
```

garden

- length
  - ?
- width
  - ?

pool

- length
  - 50
- width
  - 15

main()
1. **Abstract Data Type (ADT):** Programmer-defined data type that can be manipulated in a manner similar to the built-in data types

   ```cpp
   Rectangle pool(50, 15); // Initialize a Rectangle
   cin >> pool; // Input a Rectangle
   cout << pool; // Output a Rectangle
   if (pool > 500) // Compare a Rectangle
   ...
   ```

   Programmer who uses an ADT does not have access to details of how the values and operations are implemented.

2. **class** translates an ADT to code

   **Encapsulation:** Combines data representation and methods for manipulating that data into one package (capsule)

3. **class** specification has two parts:
   a. **Class Declaration or class prototype:** member functions (public interface) and data members

      General form:
      ```cpp
      class Class name
      {
      public:
      prototypes of member functions
      private:
      data member declarations
      }
      }; // note semicolon
      ```

      Common convention of capitalizing class names

      Similar to a structure declaration, but with member functions and public and private sections

   b. **Class method definitions:**

      Declaration provides an overview.
      Definitions supply the details.
4. access specifiers:
   a. **public** class members can be directly accessed by any user of the class, that is, any function that declares one or more objects of the class
   
   b. **private** class members can only be accessed by member functions
      
      Example: The user of a class can access private data members only by using a public member function.
      
      Public member functions provide an interface between the object's private members and the program.
      
   c. **protected** access control is used with class inheritance

5. Structures in C++ have same features as classes.

   Difference is default access specifier: **private** for classes
   **public** for structures

6. Member functions:
   Similar to regular functions: header with arguments
   body
   return type

   **2 different characteristics:**
   a. In definition, use binary scope resolution operator (::) to indicate the class to which the function belongs.
      
      :: resolves the identity of the class to which a method applies
      
      Note that :: is not needed for inline functions.
   
   b. class methods can access members of the invoking object.
7. `Rectangle::show()` qualified name
   `show()` unqualified name;
   usable only in class scope;
   `::` is needed to qualify `show()` when it is used outside of the class declaration

   `show()` can access private data members, `length` and `width`, of the invoking object

   Non-member functions cannot do this.

8. Any function that is defined in the class declaration is automatically inlined. You can also inline a function that is not defined within the class declaration:

   ```
   class Rectangle
   {
   public:
   int area(); // Prototype only
   ...
   
   inline int Rectangle::area() // Separate definition
   {
       return (length * width);
   }
   
   inline function has internal linkage; known only in the file where declared
   
   In multifile a program, put the inline definition in a header file with the class declaration.
   ```
9. When you call a member function, it uses the data members of the receiver (invoking object).

   pool.set(50, 15);
   length is interpreted as pool.length

   width is interpreted as pool.width

   garden.set(30, 20);
   length is interpreted as garden.length

   width is interpreted as garden.width

10. Goal is to make using classes as similar as possible to using built-in types:
    
    a. declare and initialize an object variable
    b. use new to allocate memory for an object variable
    c. pass object as argument and return object from a function
    d. assign one object to another, with appropriate automatic type conversions
    e. use cin and cout directly with objects

11. General characteristics of member functions:
    
    a. usually shorter than regular functions because data has been validated by constructor
    b. fewer arguments because data is built into the object