What is an Object?

- Examples of real-world **objects**: person, table, computer, city, MTR ticket machine, airline flight schedule, bank account, ...
- Most objects in the real world have **attributes**. For example, the attributes of a person include the person's age, name, gender, address, birthdate, physical condition, etc.
- Most objects have **behaviors**. Behaviors are things that an object can do. For example, a person can sing, program, walk, jump, etc.
- Consider objects as intelligent beings. Objects *know* things and they *do* things. The things an object knows are its attributes. The things an object can do are its behaviors.
- Real-world objects are often extremely complicated. Simple models can be constructed in a program to represent objects in real-world system.
- Only the characteristics of the real-world system that are relevant to the problem are represented in the corresponding model.

  For example, in a library information system to maintain information on the library's borrowers, the books and borrowers are represented as objects in the system. The attributes of each borrower may be limited to the name, id, address, list of books being checked out, amount owed for overdue books.

  In a physician's office, an information system also models each person as an object but the relevant attributes are the name, age, gender, weight, height, physical condition.

Data-directed design

Design directed by the choice and representation of data structures

- Data requirements:
  - o What functions to be performed on the data
  - o What’ s the proper scope
    - Ownership?
    - How is it shared?

Data Abstraction and Objects

- Information hidding
  - o Abstraction for external use (logical view)
  - o Hidding internal details (physical view)

- Data encapsulation
  - o Protect data from unintented modifications
  - o Access control

- Abstract data types (ADT)
  - o Specification (external)
  - o What information encapsulated, access control, operations provided
  - o Implementation (hidden)

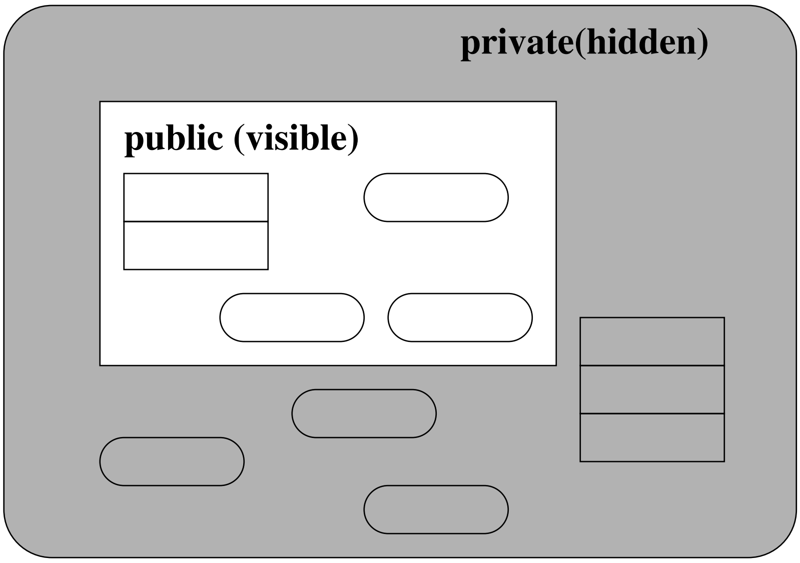
Object-Oriented Programming

Object-oriented programming is new programming paradigm.

- All data are primarily objects of abstract data types
- Objects are abstract entities.
- Internal information are hidden and encapsulated within the object; only external information and behavior is visible to outside.
- Objects have their own functions (called *methods*)
- Works are performed by invoking *methods* in an object
- Objects are categorized into types or classes
- An object can belong to a subclass of a parent class and inherit properties from its parent class.
  - o E.g. T riangle objects are inherently polygon objects.

C++ Objects – Classes

Concept



Members

- o Attributes
  - \- Data members
  - \- Data type definitions
- o Member functions

Basic access controls

- o Public
- o Private

Class – Definition Section

class *class-name*

{

public:

public-attribute-list

public-member-function-list

private:

private-attribute-list

private-member-function-list

};

Structs as Public Classes

Struct in C++ is treated as public class

```
struct employee
{
  int    ID;
  char   name[20];
  char   gender;
  int    num_depend;
  float  rate, wages;
};
```

is the same as:

```
class employee
{
  public:
    int    ID;
    char   name[20];
    char   gender;
    int    num_depend;
    float  rate, wages;
};
```

Example:

```
// FILE: counter.h
//   Definition file
#ifndef COUNTER_H
#define COUNTER_H

class counter
{
  public:
    // Member functions
    // Initialize counter
    void initialize();
    // increment counter
    void increment();
    // decrement counter
    void decrement();
    // get counter value
    int access_value();

  private:
    // Private data members
    int value;
}; // Must end with a semicolon

#endif  // COUNTER_H
```

Class – Implementation Section

type class-name **::** *fname*

(*argument-list*)

{

function-body

}

Scope resolution operator **::**

Example:

```
// FILE: counter.cpp
//   Class implementation
#include <iostream.h>
#include <limits.h>
#include "counter.h"

// initialize counter
void counter::initialize()
{
  value = 0;
}
```

```
// increment counter
void counter::increment()
{
  if (value < INT_MAX)
    value++;
  else
    cerr << "Counter overflow: "
         << "Increment ignored."
         << endl;
}


// decrement counter
void counter::decrement()
{
  if (value > INT_MIN)
    value--;
  else
    cerr << "Counter underflow: "
         << "Decrement ignored."
         << endl;
}


// get counter value
int counter::access_value()
{
  return value;
}
```

Declaring class objects

class-name *variable-name* ;

Accessing class members

class-variable-name **.***member-name*

```
#include "counter.h"
  counter cntr;
  int current_val;
  cntr.initialize();
  cntr.increment();
  cntr.decrement();
  current_val = cntr.value;  // NOT ALLOWED!
  current_val = cntr.access_value();
```

Example:

```
// Create a class of checking accounts
class checking_account
{
  public:
    // Member functions
    // Initialize balance
    void initialize();

    // Deposit into account
    void deposit( int );       // IN: account number

    // Withdraw from account
    void withdraw ( int );     // IN: account number

    // get account balance
    int get_balance ( int );   // IN: account number

  private:
    // Private data members
    int account_number;
    float balance;
};


// Deposit into account
void checking_account::deposit (int acc_no ) {
    float amount;

    if (acc_no == account_number) {
      cout << "Get amount to be deposited: " <<endl;
      cin >> amount;
      balance += amount;
    }
    else {
      cout << "Wrong account number" << endl;
    }
    return;
}
```

Object Creation and Destruction

A **constructor** is a member function that is automatically called when a new object of that class type is created.

- A constructor has no return type
- A constructor has the same name as the class itself.

Constructor

class-name **::** *class-name* ();

Destructor

class-name **::** ~ *class-name* ();

```
// counter constructor
counter::counter()
{
  value = 0;
}
// counter destructor
counter::~counter()
{
  // any clean up work
}
```

A Complex Number Class Example

```
// FILE: complex.h
//   Defines complex number class

class complex
{
  public:
    // constructors
    complex();
    complex(float, float);
    // get real and imaginary part
    float get_real();
    float get_im();
    // operations
    complex add(const complex&);
    complex multiply(const complex&);
    ...
  private:
    float real;
    float im;
};
```

```
// FILE: complex.cpp
//   Complex number class implementation
#include "complex.h"
// Constructors
complex::complex() {}
complex::complex(float x, float y)
{
  real = x;
  im = y;
}
// get real and imaginary part
float complex::get_real()
{
  return real;
}
float complex::get_im()
{
  return im;
}
// Operations
complex complex::add(const complex& x)
{
  complex result;
  result.real = real + x.real;
  result.im = im + x.im;
  return result;
}
complex complex::multiply(const complex& x)
{
  complex result;
  result.real = real*x.real - im*x.im;
  result.im = real*x.im + im*x.real;
  return result;
}
```

Example:

```
#include "complex.h"
int main()
{
   complex x(2.0, 1.0);
   complex y(1.0, -3.0);
   complex z1, z2;
   z1 = x.add(y);
   z2 = x.multiply(y);

   return 0;
}
```