CS2311 Computer Programming

LT12: Class & Object

Outline

- Defining classes
- Defining member functions & scope resolution operator
- Public & private members
- Constructors
- Accessors

Class and Object

- Class and object are important features of Object-oriented Programming Language (C++, Java, C#)
- With class, variables and their directly related functions can be grouped together to form a new data type
- It promotes reusability and object-oriented design (not covered in this course)
- Object is an instance of class, i.e. *class* is a blue-print and its product is its *object*.

Class and Object : Example

Without class/object

```
int radius;
int width, height;

double getCircleArea( ) {
   return   3.14*radius*radius;
}
double getRectangleArea() {
   return  width*height;
}
double getCirclePerimeter() {
   return   2*3.14*radius;
}
double getRectanglePerimeter()
   {
   return  2*(width+height);
}
```

With class/object

```
class Circle {
public:
   int radius;
   double getArea( ) {
       return 3.14*radius*radius;
   }
   double getPerimeter() {
       return 2*3.14*radius;
   }
};
```

```
class Rect {
public:
   int width, height;
   double getArea() {
     return width*height;
   }
   double getPerimeter() {
     return 2*(width+height);
   }
};
```

Class and Object

Without class/object

```
void main(){
   cout << "Please enter the radius of circle";
   cin >> radius;
   cout << getCircleArea();

   cout << "Please enter the width and height of a rectangle";
   cin >> width >> height;
   cout << getRectangleArea();
}
```

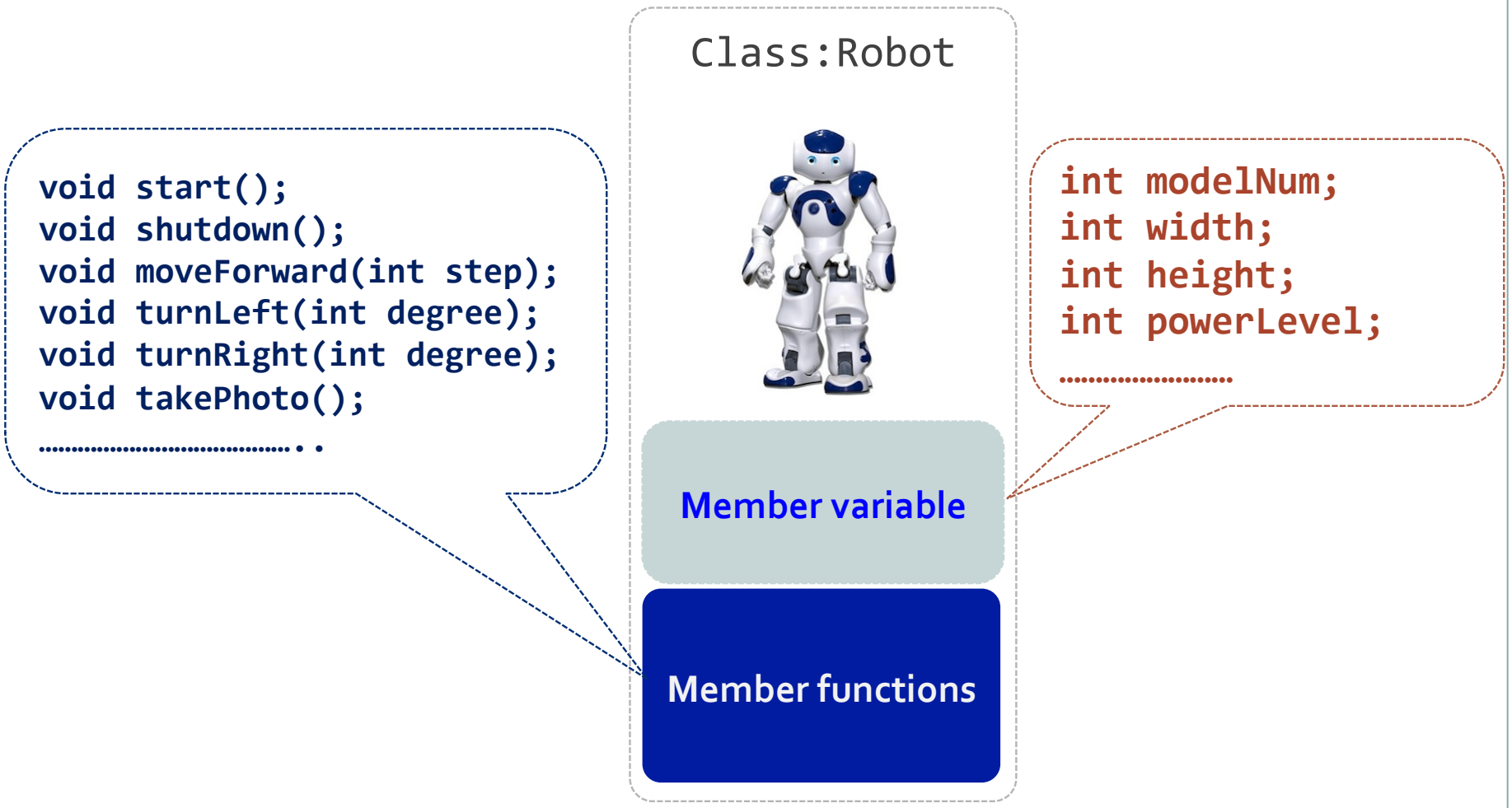
With class/object

```
int main() {
   Rect r;   // Rect is a class, r is an object of Rect
   Circle c;
   cout << "Please enter the radius of circle";
   cin >> c.radius;
   cout << c.getArea();
   cout << "Please enter the width and height of a rectangle";
   cin >> r.width >> r.height;
   cout << r.getArea();
   return 0;
}
```

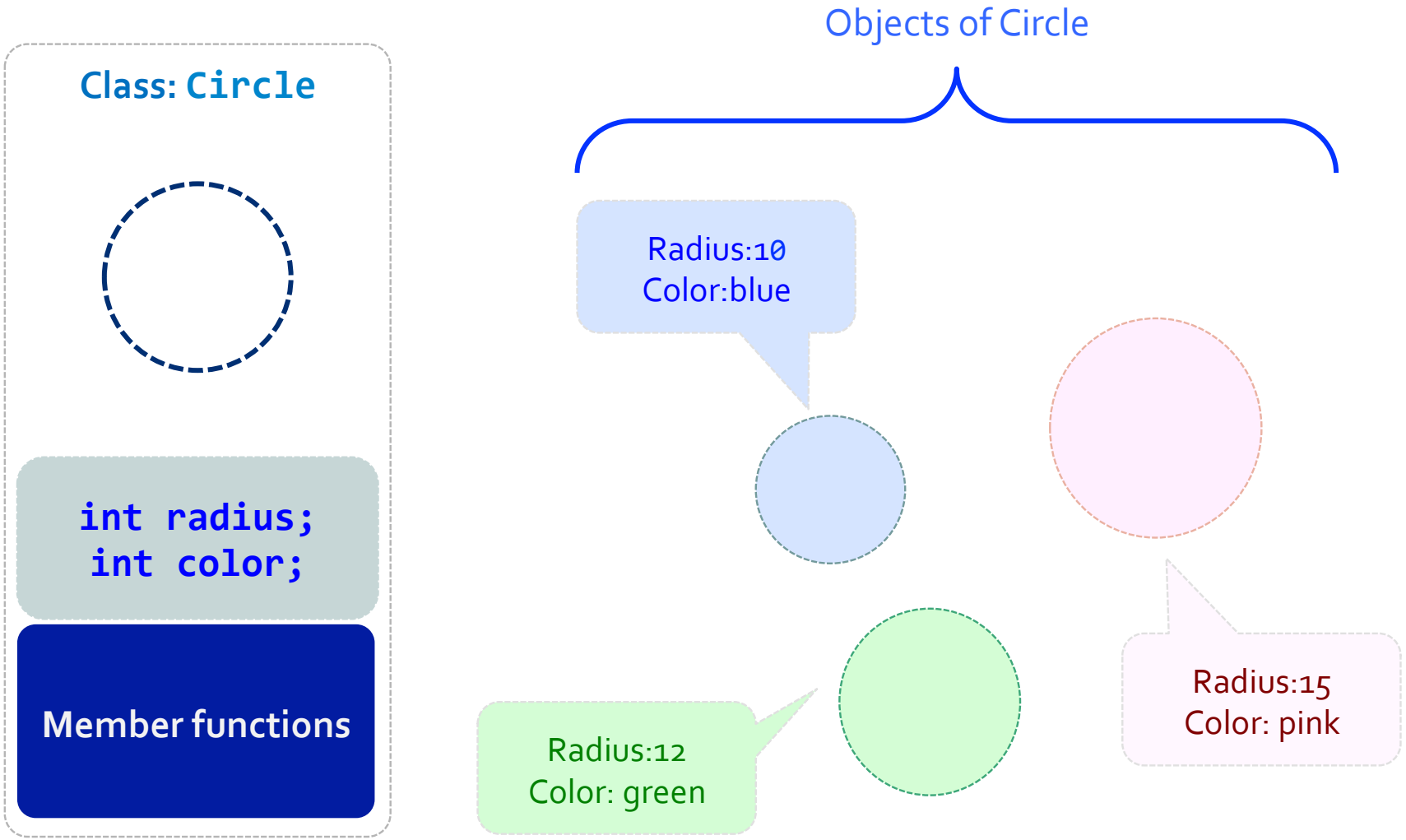
Class in Computer Programming

- An abstract view of real-world objects, e.g. car, horse
- Computer program is a model of real-world problem
- Simple problem: program with variables and functions
- Large scale program: class and object
- **Class:**
  - definition of program component
  - consists of member variables and member functions
  - Member variable : variable belong to class
  - Member function: function primary designed to access/ manipulate the member variable of the class
- **Object:**
  - An instance of class / runtime representation of a class

Class in programming



What is an object?



Classes and Objects in C++

- A class is a data type, objects are variables of this type
- An object is a variable with member functions and data values
- **`cin`**, **`cout`** are objects defined in header **`<iostream>`**
- C++ has great facilities for you to define your own class and objects

Defining `Class`

Key concept:

Encapsulation

```
class class_name {
   public / protected / private:
      attribute1 declaration;
      attribute2 declaration;
      method1 declaration;
      method2 prototype;
};
return_value classname::method2(args) {
      method body statement;
}
```

Defining `Class` Example 1

```
#include <iostream>
using namespace std;
class DayOfYear {
  public:
     int month;
     int day;
     void output() {
        cout << "month = " << month;
        cout << ", day = " << day << endl;
     }
};
```

Member variables

Member method/function

Member Function

- In C++, a class definition commonly contains only the prototypes of its member functions (except for inline functions)
- Use **`classname::functionName`** to define the member function (method) of particular class.

```
class Circle {
  …
  int radius;
  …
  double getArea();
};
double Circle::getArea() {
  return 3.1415*radius*radius;
}
```

Defining `Class` Example 2

```
#include <iostream>
using namespace std;
class DayOfYear {
public:
  void output(); //member func. prototype
  int month;
  int day;
};
void DayOfYear::output() {
  cout << "month =" << month << ", day =" <<
  day << endl;
}
```

Separation of
class Interface
from
class Implementation

Define the method elsewhere

Create Object: Access Member Function

- To declare an object of a class

```
Class_name   variable_name;
```

Examples:

```
Circle c1,c2;
DayofYear today;
```

- A member function of an object is called using the dot operator:

```
today.output();
c1.getArea();
```

Create Pointer: Access Member Function

- We can also declare the pointer of a class

```
Class_name*    pointer_name;
```

Examples:

```
Circle *c1, *c2;
DayofYear *today;
```

- A member function of the pointer is called using the arrow operator (->):

```
today->output();
c1->getArea();
```

Test Driver `main` Function

```
int main() {

    DayofYear   today, birthday;
    cin >> today.month >> today.day;
    cin >> birthday.month >> birthday.day;
    cout << "Today's date is: ";
    today.output();
    cout << "Your birthday is: ";
    birthday.output();

    if (today.month == birthday.month
       && today.day == birthday.day)
       cout << "Happy Birthday!" << endl;
    return 0;
}
```

`public` and `private` Members

- By default, all members of a class are **`private`**
- You can declare public members using the keyword **`public`**
- **`private`** members can be accessed only by member functions (and **`friend`** functions) of that class, i.e. only from within the class, not from outside

Private Variables and Access Functions

- Member functions that give you access to the values of the private member variables are called access functions, e.g.,

  `get_month, set`

Key concept:

Encapsulation

- Useful for controlling access to private members:
  - E.g. Provide data validation to ensure data integrity.

A Class Definition for DayOfYear

```
class DayOfYear {
public:
   DayOfYear();  // default constructor
   DayOfYear(int m, int d); // initialization constructor
   DayOfYear(const DayOfYear& doy); // copy constructor
   void input();
   void output() const;
   bool isSame(const DayOfYear& doy) const;
private:
   int month;
   int day;
};
```

Constructors for initialization

- Class contains variables and functions
- Variables should be initialized before use in many cases
- In C++, a constructor is designed to initialize variables
- A ***constructor*** is a member function that is automatically called when an object of that class is declared
- Special rules:
  - A constructor must have the same name as the class
  - A constructor definition cannot return a value

Default Constructor

```
// default constructor: Jan 1
DayOfYear::DayOfYear() {
   month = 1;
   day = 1;
}
```

Initialization Constructor

```
// initialization constructor
DayOfYear::DayOfYear(int m, int d) {
    if (valid(m,d)) {
        month = m;
        day = d;
    }
    else { // exception handling
         cout << "Invalid input." << endl;
    }
}
```

Copy Constructor

```
// copy constructor
DayOfYear::DayOfYear(const DayOfYear& doy) {
   month = doy.month;
   day = doy.day;
}
```

pass by reference

parameter should not be modified

Defining `Class` Member Functions

```
void DayOfYear::output() const {
  cout << "month = " << month << ", day = " << day <<
  endl;
}

bool DayOfYear::isSame(const DayOfYear& doy) const {
  return (month == doy.month && day == doy.day);
}
```

function canNOT modify member's value

Member Function Definition

```
void DayOfYear::input() {
   int m, d;
   // input and validate
   do {
      cin >> m >> d;  // local var. of input()
   } while (!valid(m,d));
   month = m;  // accessing private members
   day = d;
}
```

Inline Function

```
inline bool valid(int m, int d) {

    if (m<1 || m>12 || d<1)
        return false;
    switch(m) {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            return d <= 31; break;
    case 4: case 6: case 9: case 11:
            return d <= 30; break;
    case 2:
            return d <= 29; break;
    default:
            return true;
    }
}
```

Main Program

```
int main() {
    DayOfYear today;
    DayOfYear birthday(10,1);
    DayOfYear myday(birthday);
    cout << "Today's date is: ";
    today.output();
    if (myday.isSame(birthday))
        cout << "Happy Birthday!" << endl;

    cout << "Today: ";
    today.input();
    cout << "Birthday: ";
    birthday.input();
    cout << "Today's date is: ";
    today.output();
    cout << "Your birthday is: ";
    birthday.output();
    if (today.isSame(birthday))
        cout << "Happy Birthday!" << endl;
    return 0;
}
```

Private Variables and Access Functions

- Member functions that give you access to the values of the private member variables are called access functions, e.g., `get_month, set`

Key concept:

Encapsulation

- Useful for controlling access to private members:
  - E.g. Provide data validation to ensure data integrity.

- Needed when testing equality of 2 objects. (The predefined equality operator `==` does not work for objects and variables of structure type.), e.g. `obj1 == obj2` (not working!)

Why `private` Member?

- Prevent others from accessing the members directly, i.e. members can be only accessed by member functions.

```
class DayOfYear {
    ……
private:
    int month;
    int day;
    ……
};
```

```
void DayOfYear::set(int new_m, int new_d) {
    month = new_m;
    day   = mew_d;
    ……
}
int DayOfYear::get_month() const {
    return month;
}
int DayOfYear::get_day() const {
    return day;
}
```

Why `private` variable?

- Change of the internal presentation, e.g. variable name, type, will not affect the how the others access the object. Caller still calling the same function with same parameters

```
class DayOfYear {
    ……
private:
    int month;
    int day;
    ……
};
```

```
void DayOfYear::set(int new_m, int new_d) {
    month = new_m;
    day   = mew_d;
    ……
}
int DayOfYear::get_month() const {
    return month;
}
int DayOfYear::get_day() const {
    return day;
}
```

Why `private` members?

- The common style of class definitions
  - To have all member variables **`private`**
  - Provide enough access functions to get and set the member variables
  - Supporting functions used by the member functions should also be made **`private`**
  - Only functions that need to interact with the outside can be made **`public`**

Assignment operator for objects

- It is legal to use assignment operator = with objects
- E.g.

```
DayOfYear due_date, tomorrow;
tomorrow.input();
due_date = tomorrow;
```

- This effectively makes both variables pointing to the same memory address of the object
  - for class with simple data members, the operator = may work
  - avoid this assumption
- If a copy constructor is included, the operator = may work
- We may implement an *operator overloading function* for =

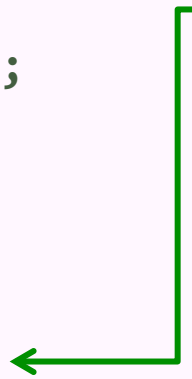
Default Constructor

- A constructor with no parameters
- Will be called when no argument is given

```
class Circle {
public :
    Circle();
    double getArea() const;
private :
    int radius;
};

void Circle::Circle() {
    radius = 0;
}
double Circle::getArea() const {
    return 3.1415*radius;
}
```

```
int main() {
    Circle circle;
    cout << circle.getArea();
    return 0;
}
```



Default Constructor & Initilization Constructor

- A default constructor will be generated by compiler automatically if NO constructor is defined.
- However, if any non-default constructor is defined, calling the default constructor will have compilation error.

```
class Circle {
public :
    Circle(int r);
    double getArea();
private :
    int radius;
};
void Circle::Circle(int r) {
    radius = r;
}
double Circle::getArea() const {
    return 3.1415*radius;
}
```

```
int main() {
✗ Circle circle;     // illegal
  Circle circle(6); //OK
  cout << circle.getArea();
  return 0;
}
```

Constructor Initialization

- A constructor with no parameters
- Will be called when no argument is given

```
class Circle {
public :
    Circle(): radius(0) {}
    Circle(int r): radius(r) {}
    double getArea() const;
private :
    int radius;
};
double Circle::getArea() const{
    return 3.1415*radius;
}
```

```
int main() {
    Circle circle1;
    Circle circle2(8);
    cout << circle2.getArea();
    return 0;
}
```

member function prototype

member function definition/ implementation

Example: Bank account

- E.g., Suppose we want to define a bank account class which has member variables **balance** and **interest_rate**. We want to have a constructor that initializes the member variables.

```
class BankAcc {
public:
    BankAcc(int dollars, int cents, double rate);
  ...
private:
    double balance;
    double interest_rate;
};
...
BankAcc::BankAcc(int dollars, int cents, double rate) {
     balance = dollars + 0.01*cents;
    interest_rate = rate;
}
```

Constructors

- When declaring **BankAcc** objects:

```
BankAcc  account1(10,50,2.0),
         account2(500,0,4.5);
```

Note:
A constructor cannot be called in the same way as an ordinary member function is called:

```
account1.BankAcc(10,20,1.0); // illegal
```

Constructors

- More than one version of constructors are usually defined (overloaded) so that objects can be initialized in more than one way, e.g.

```
class BankAcc   {
public:
    BankAcc(int dollars, int cents, double rate);
    BankAcc(int dollars, double rate);
    BankAcc();
     ...
private:
    double balance;
    double interest_rate;
};
```

Constructors

```
BankAcc::BankAcc(int dollars, int cents, double rate) {
    balance = dollars + 0.01*cents;
    interest_rate = rate;
}
BankAcc::BankAcc(int dollars, double rate) {
    balance = dollars;
    interest_rate = rate;
}
BankAcc::BankAcc() {
    balance = 0;
    interest_rate = 0.0;
}
```

Constructors

- When the constructor has no arguments, don't include any parentheses in the object declaration.
- E.g.

```
BankAcc  acc1(100, 50, 2.0),  // OK
         acc2(100, 2.3),      // OK
         acc3(),              // error
         acc4;                // correct
```

- The compiler thinks that it is the prototype of a function called **acc3** that takes no argument and returns a value of type **BankAcc**

Copy Constructor

```
class Circle {
public :
    Circle();
    Circle(int r);
    Circle(const Circle& c);
    double getArea() const;
private :
    int radius;
};
Circle::Circle() {
    radius = 0;
}
Circle::Circle(int r) {
    radius = r;
}
Circle::Circle(const Circle& c) {
    radius = c.radius;
}
double Circle::getArea() const{
    return 3.1415*radius;
}
```

```
int main() {
    Circle circle1;
    Circle circle2(8);
    Circle circle3(circle2);
    cout << circle3.getArea();
    return 0;
}
```

Copy Constructor

- C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects
  - the compiler created copy constructor works fine **in general**
- Need to define our own copy constructor when an object has **pointers** or any ***runtime allocation*** of the resource
- Alternative way to call a copy constructor:

```
obj = constr_name(arguments);
```

E.g.,
```
BankAcc account1;
account1 = BankAcc(200, 3.5);
```

- Mechanism: calling the constructor creates an anonymous object with new values; the object is then assigned to the named object
- A constructor behaves like a function that returns an object of its class type

Summary

- Object-oriented programming is the main trend.
- Encapsulation concept
- Information hiding is a significant design principle