WEEK 6
OBJECTS AND CLASSES

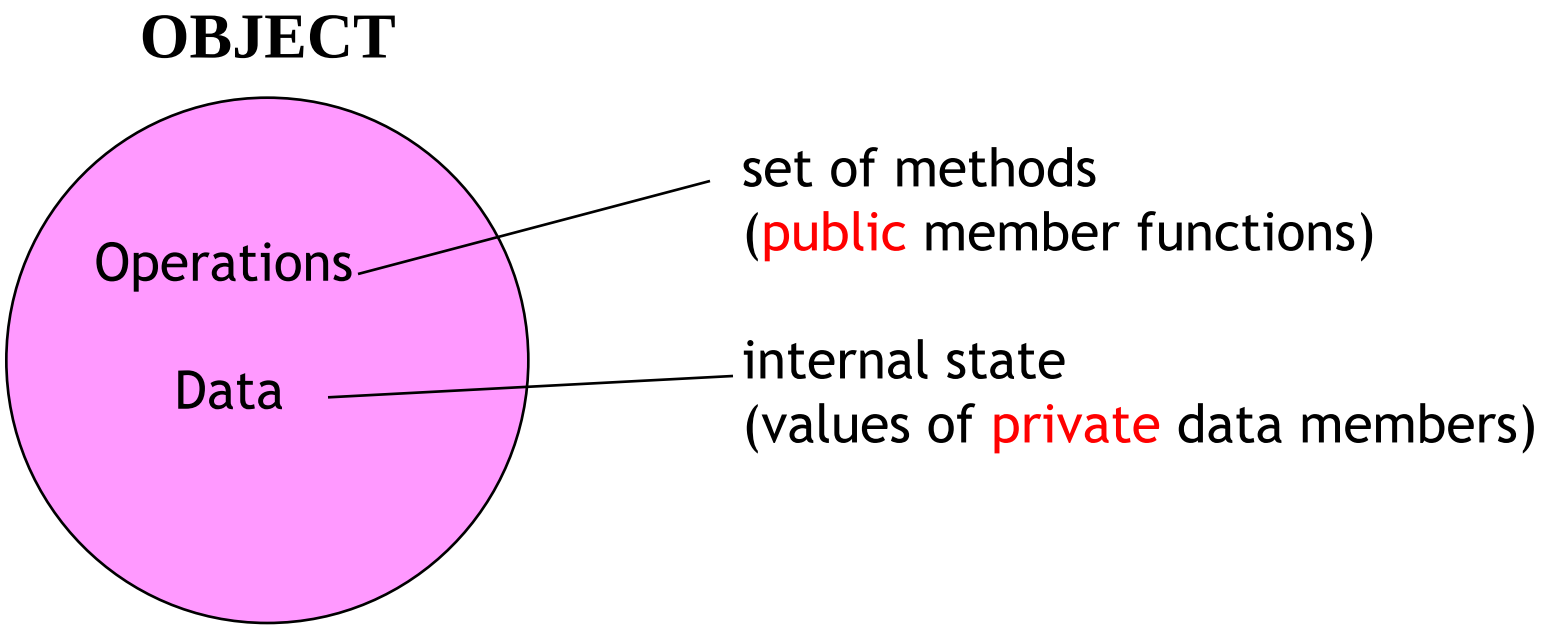
Dr. Yuzana Win

OUTLINE

- What is an Object?
- Defining the Class
- Access Control Specifiers
- Constant Functions
- Static vs. Non-static Members
- Constructors and Destructors

WHAT IS AN OBJECT?

- Variables or instances of a class that is declared in a program



DEFINING THE CLASS

- **Class**
  - Consists of both data and functions/methods
  - Defines properties and behavior of set of entities
  - Defines a new type (like int, float etc.)

- **Object**
  - ☑ An instance of class
  - ☑ Has identity, state, and behavior

SYNTAX OF CLASS

Syntax of class

class *class_name* {
    access_specifier_1:
        member1;
    access_specifier_2:
        member2;
} *object_names*;

```
class age
{
    private:
        int age1;
    public:
        void getage()
        {
           cin >> age1;
        }
};
```

CLASS INCLUDES:

- **Data Members** (The data item within a class)
- **Member Functions** (Functions that are included within a class)

- Access Control Specifiers

- Constant functions

- Static vs. Non-static members

- Constructors & Destructors

ACCESS CONTROL SPECIFIERS

- **Public**
  - Can be accessible from anywhere in the program
- **Private (default)**
  - Can be accessible only from member functions of its class and friends
- **Protected**
  - Acts as public for objects of its own class and derived classes
  - Acts as private to rest of the program
- ***Public functions*** – Class Interface
- ***Private functions*** – helper functions (can be accessed by class objects and friends)

EXAMPLE 1

```
// classes example
#include <iostream>
using namespace std;
class  Rectangle
{
   private:
          int x, y;
   public:
          void set_values (int a, int b)
          { x = a; y = b; }
          int area ()
          {
              return (x*y);
          }
};
```

```
int main ()
    {
          Rectangle rect;
          rect.set_values (3,4);
          cout << "area: " <<
         rect.area();
          return 0;
    }
```

Output: area: 12

EXAMPLE 2

```
// classes example
#include <iostream>
using namespace std;
class  Rectangle
{
   private:
          int x, y;
   public:
          void set_values (int, int);
          int area ()
          {
                  return (x*y);
          }
};
```

```
void  Rectangle::set_values (int a,
    int b)
{          x = a;
          y = b;
}
int main ()
     {
          Rectangle rect;
          rect.set_values (3,4);
          cout << "area: " <<
    rect.area();
          return 0;          }
```

Output: area: 12

EXAMPLE 3

```
// classes example
#include <iostream>
using namespace std;
class  Rectangle
{
   private:
          int x, y;
   public:
          void get_values ()
          {
            cout << “Enter the first
   value\n”;
            cin >> x;
            cout << “Enter the
   second value\n”;
             cin >> y;
          }
```

```
int area ()
{
          return (x*y);
}
};
int main ()
     {
          Rectangle rect;
          rect.get_values ();
          cout << "area: " <<
     rect.area();
          return 0;          }
```

Output: area: ???

CONSTANT MEMBER FUNCTIONS

- Constant Member Function
  - Does not modify the state of the object !

- Const function can be invoked from both const & non-const objects
- Non-const function can be invoked only from non-const object

- **Declaration**

  return_type **func_name** (*para_list*) const;

- **Definition**

  return_type **func_name** (*para_list*) const { … }

EXAMPLE OF CONST MEMBER FUNCTION

```
class  Time
{
   private :
       int  hrs, mins, secs ;

   public :

       void  Write ( )  const ;

} ;
```

function declaration

function definition

```
void Time :: Write( ) const
{
    cout <<hrs << ":" << mins << ":" << secs << endl;
}
```

STATIC VS. NON STATIC DATA MEMBERS

- Abstracts the general attributes of the entity defined by class.
- Type can be anything: *built-in* or *user-defined*.

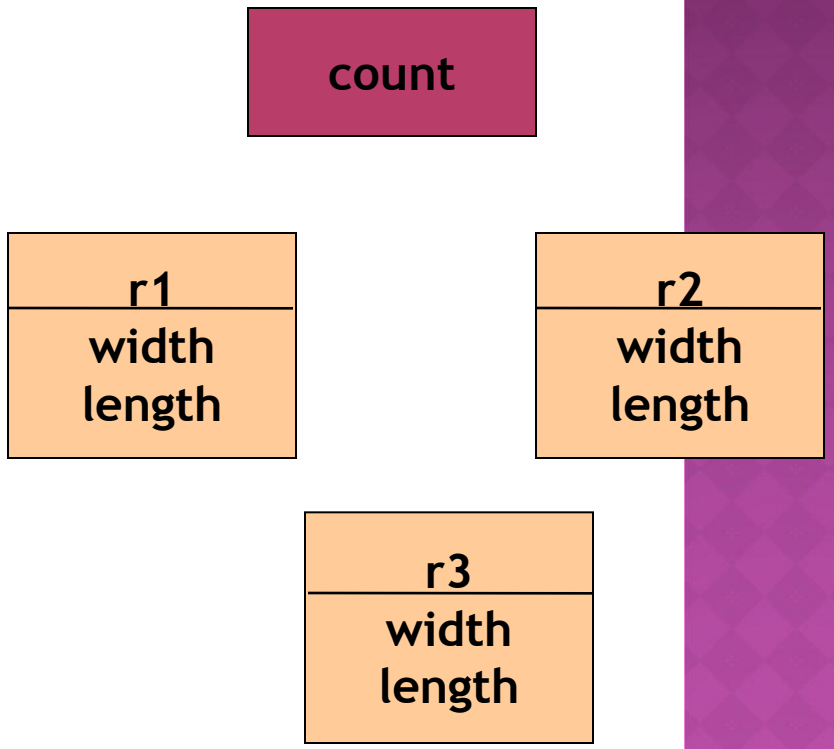
	Non-Static Member	Static Member
Existence	***Local*** to object – each object has its own copy	***Global*** to class – all objects *share* the *same copy*
Initialization	object level in member functions or constructor	Class level
Access	Qualified by object <object name>.<var>	Qualified by class <class name>::<var>

STATIC DATA MEMBER (CLASS LEVEL)

```
class Rectangle
{
    private:
        int width;
        int length;
        static int count;

    public:
        void set(int w, int l);
        int compute_area();
}
```

```
Rectangle    r1;
Rectangle    r2;
Rectangle    r3;
```



STATIC VS. NON STATIC MEMBER FUNCTIONS

- **Non-static Member Function**
  - Invoked from objects [*object.function()*]
  - Can access both *static* and *non-static* data members

- **Static Member Function**
  - Invoked using class [classname::function()]
  - Can only access static data members (but not non-static)

CONSTRUCTOR AND DESTRUCTOR

Constructor and Destructor

- Constructor and Destructor are special member functions of class that are used to construct and destroy the class's object.

- Constructor are called every time you created an object. So Constructor are **initialization function** of object.

- Destructor are called every time you destroy an object.

Constructor and Destructor

	Constructor	Destructor
Function Name	Same with Class Name	~ with Same Class Name
Return Type	No return and No need to declare return	No return and No need to declare return
Parameter Passing	It can have different number of parameters and it can overload	No need for parameter

OBJECT INITIALIZATION WITH DIFFERENT TYPE OF CONSTRUCTOR

- An object can be initialized by a class constructor
  - default constructor
  - constructor with parameters
  - copy constructor

DEFAULT CONSTRUCTOR

```
class Example {
        public:
                int a,b,c;

                void multiply (int n, int m)
                {
                        a=n;
                        b=m;
                        c=a*b; };
};
```

- ✓ *Object declaration :* **Example ex;**
- ✘ **Example ex();**

INITIALIZATION WITH DEFAULT CONSTRUCTOR

```
class Rectangle
{           private:
               int width, length;
            public:
               Rectangle ( )
               { width = 0;
                 length = 0;
               }
                 void display()
                  { cout<<
width<<endl;
                      cout<<length;
                  }
};
```

Class

Object

```
main()
{
     Rectangle rect;
     rect.display();
}
```

Initialization of Object using default Constructor

Output
0
0

Initialization with constructor with parameter

```
class Rectangle
{private:
      int width, length;
  public:
    Rectangle (int w, int l)
    {  width = w;
         length = l;
    }
      void display()
      { cout<< width<<endl;
        cout<<length;
      }
};
```

Class

Object

```
main()
{
     Rectangle rect(3,5);
      rect.display();
}
```

Initialization of Object using constructor with parameter

Output

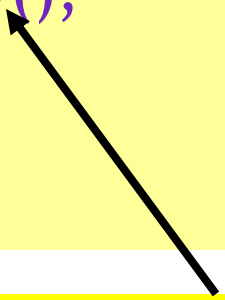
3

5

Initialization with copy constructor

```
class Rectangle
{
  private:
       int width, length;
  public:
       Rectangle (int w, int l)
      {
         width = w;  length = l;
        }
        Rectangle (Rectangle& r)
        {   width  = r.width;
         length = r.length;
         }
         void display()
        { cout<< width<<endl;
        cout<<length;
        }
};
```

```
int main()
{ Rectangle r4(60,80);
   Rectangle r5(r4);
    r5.display();
return 0;
 }
```



Initialization using copy Constructor

Output
60
80

CONSTRUCTOR OVERLOADING

- a constructor can also be overloaded with more than one function that have the **same name** but **different types or number of parameters**.

- automatically called when an object is created, the one executed is the one that matches the arguments passed on the object declaration.

CONSTRUCTOR OVERLOADING

```
#include <iostream>
using namespace std;
class Rectangle
 {
    private:
          int width, height;
    public:
          Rectangle ();
          Rectangle (int, int);
          int area ()
            {
              return (width*height);
            }
};
Rectangle::Rectangle () {
       width = 5;
       height = 5;
}
```

```
Rectangle::Rectangle (int a, int b) {
       width = a;
       height = b;
}

int main () {
       Rectangle rect (3,4);
       Rectangle rectb;
       cout << "rect area: " <<
          rect.area()  << endl;
        cout << "rectb area: " <<
          rectb.area() << endl;
      return 0;
}
```

What will be the output? ➔ ???

POINTERS TO CLASSES

- can use the class name as the type for the pointer.

Example:

Rectangle * prect;

is a pointer to an object of class Rectangle.

- use the arrow operator (->) of indirection, in order to refer directly to a member of an object pointed by a pointer.

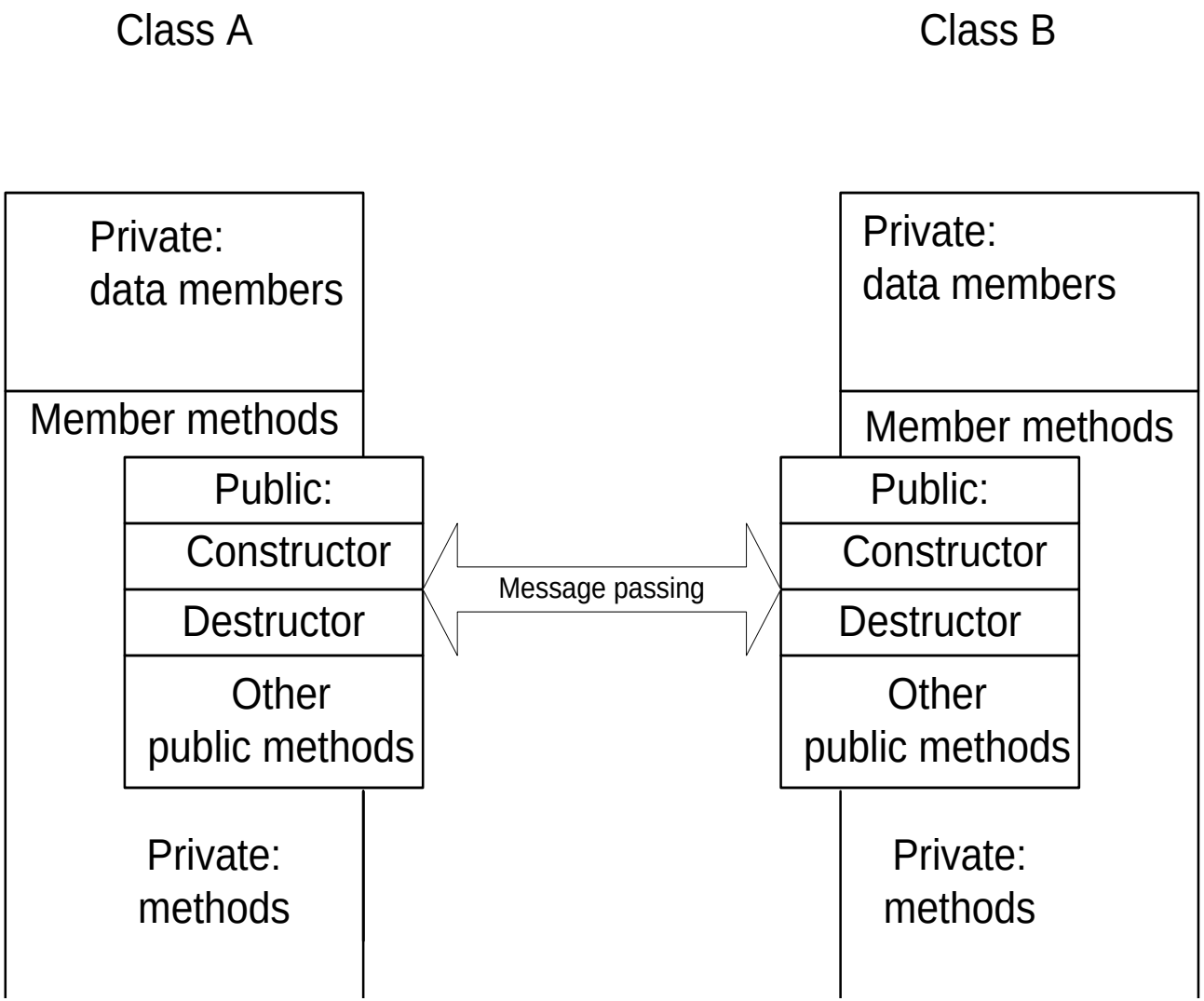
POINTER TO CLASSES

```
#include <iostream>
using namespace std;
class Rectangle {
        private:
                int width, height;
        public:
                void set_values (int,
int);
                int area ()
                {return (width *
height);}
};
void  Rectangle::set_values (int a,
int b)
 {
      width = a;
      height = b;
}
```

```
int main () {
   Rectangle a, *b, *c;
   Rectangle * d = new Rectangle[2];
   b= new Rectangle();
   c= &a;
   a.set_values (1,2);
   b->set_values (3,4);
   d->set_values (5,6);
   d[1].set_values (7,8);
   cout << "a area: " << a.area() << endl;
   cout << "*b area: " << b->area() << endl;
   cout << "*c area: " << c->area() << endl;
cout << "d[0] area: " << d[0].area() <<
endl;
cout << "d[1] area: " << d[1].area() <<
endl;
return 0;
}
```

What will be the output? ➔ ???

INTERACTING OBJECTS



EXERCISES

1. Create a class that contains four member functions, with 0, 1, 2 and 3 int arguments, respectively.

Create a main() that makes an object of that class and calls each of the member functions.

Now Modify the class so it has instead a single member function with all the arguments defaulted.

Does this change the main()?

EXERCISES

2. Create a class called **time** that has separate int member data for hours, minutes, and seconds.

- One constructor should initialize this data to 0, and another should initialize it to fixed values.

- Another member function should display it, in 11:59:59 format.

- The final member function should add two objects of type time passed as arguments.

A main () program should create two initialized time objects and one that isn't initialized.

Then it should add the two initialized value together, leaving the result in the third time variable.

Finally it should display the value of this third variable. Make appropriate member functions const.

EXERCISES

3. Create a class *Distance* which includes two data members: *feet* and *inches*, two constructors: default constructor and parameter constructor and three member functions: *getdist*, *showdist* and *add_dist* two distances.

Create three objects in the main: two objects *dist1* and *dist3* by no argument constructor and another object *dist2* by parameter constructor.

Update the *dist1* value by calling member function *getdist*. Then add two objects *dist1* and *dist2* and assign it to *dist3*.

Show the data in *dist1*, *dist2* and *dist3* by calling *showdist* function.