Hochiminh City University of Technology

Computer Science and Engineering

[CO1011 - 501127] - Fundamentals of C++ Programming

Class

Lecturer: Dr. Duc Dung Nguyen
Dr. Rang Nguyen
Dr. Phung Nguyen

Credits: 4

Outcomes

- Explain some basic concepts of Class:
  - Encapsulation
  - Data hiding
  - Class member: fields, methods
  - Class access modifiers
  - Static and instance member
  - Constructor and destructor

Outline

- Class:
  - Concept and definition
  - Encapsulation
- Constructor / Destructor
- Other issues

Data Types

- **Scalar:** Integer (int), Float (float), Double (double), Char (char)
- **Structured:** Array (int[], char[],…), Struct (struct), File

=> Variables of these types just keep **data only**.

```
struct Rectangle {
    double width;
    double height;
}
```

Class

Class

- Class: a datatype which groups together related pieces of information
    - Data: Fields (Variable, Constant)
    - Behaviours: Methods (Functions)
- **Classes** are similar to **Structure** but contain functions, as well.

Class Example

```
class Rectangle
{
private:
   double width;
   double height;
public:
   void setWidth(double);
   void setHeight(double);
   double getWidth();
   double getHeight();
   double getArea();
};
```

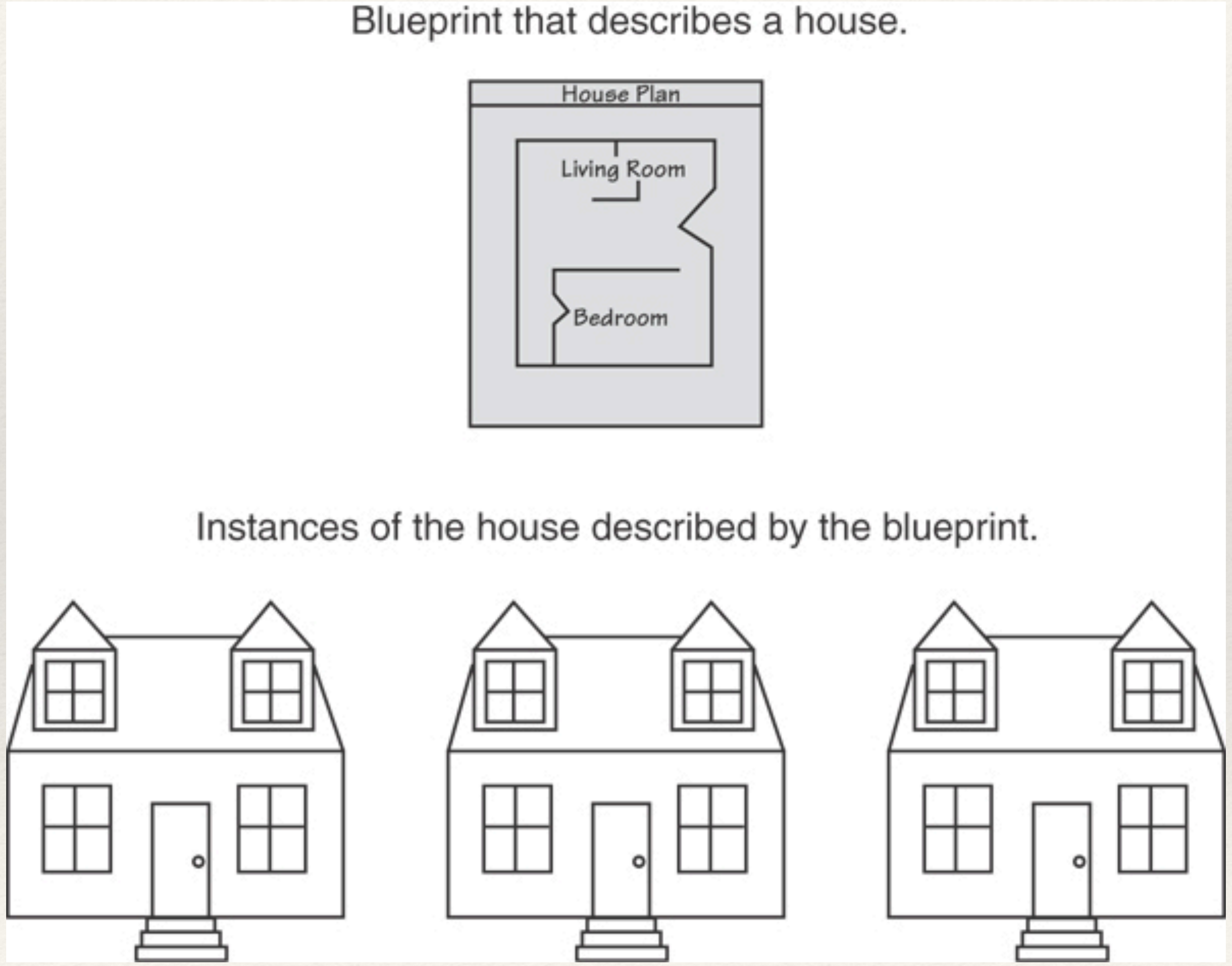
Fields

Methods

```
struct Rectangle {
    double width;
    double height;
};
```

Classes and Objects

- A Class is like a blueprint and Objects are like houses built from the blueprint


Blueprint that describes a house.
House Plan
Living Room
Bedroom
Instances of the house described by the blueprint.

Objects Example

Class

```
class Rectangle
{
private:
   double width;
   double height;
public:
   void setWidth(double);
   void setHeight(double);
   double getWidth();
   double getHeight();
   double getArea();
};
```

Objects

width:30; height: 20;
setWidth(double);
setHeight(double);…

Width:15; height: 10;
setWidth(double);
setHeight(double);…

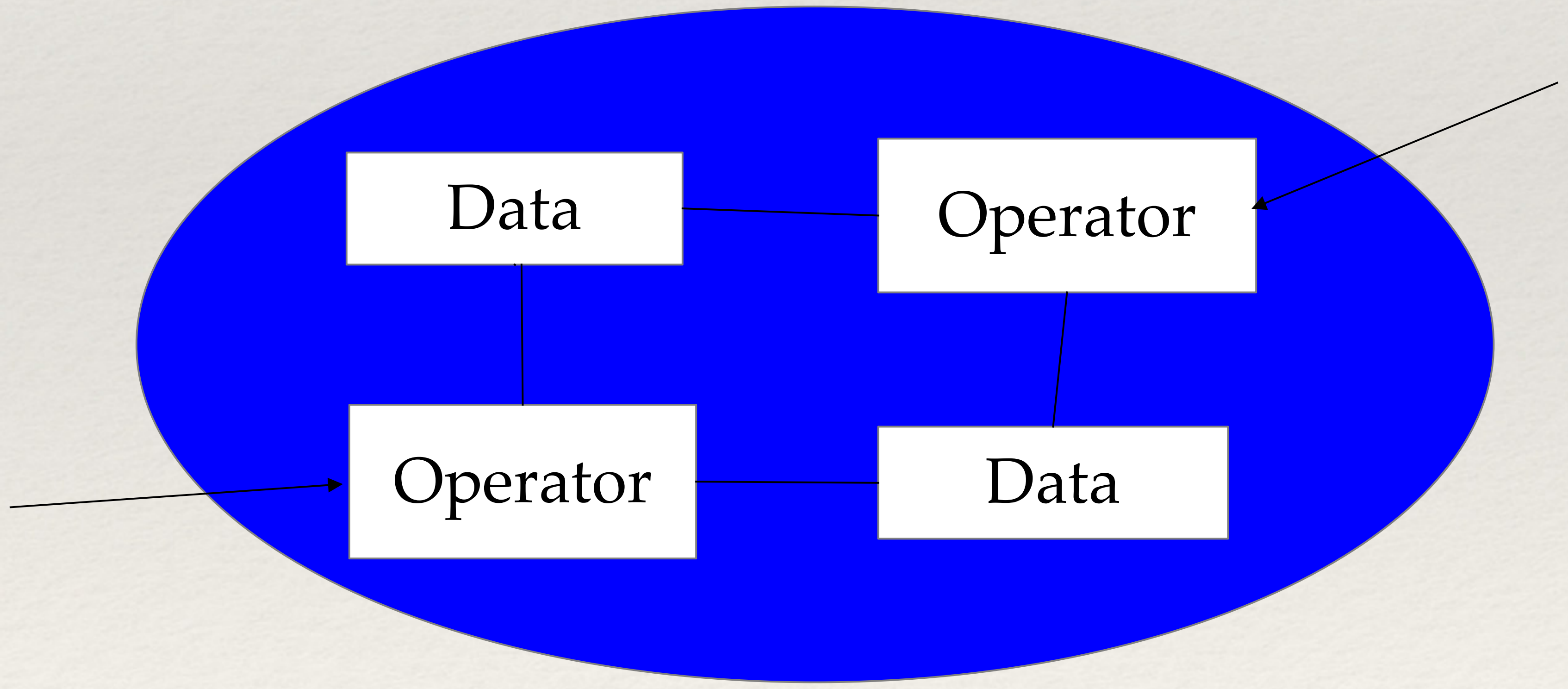
width:25; height: 15;
setWidth(double);
setHeight(double);…

Features

- Encapsulation (hiding data): allows the programmer to group data and the behaviours that operate on them together in one place, and to hide irrelevant details from the user.
- Inheritance: allows code to be reused between related types.
- Polymorphism: allows a value to be one of several types, and determining at runtime which functions to call on based on its type.

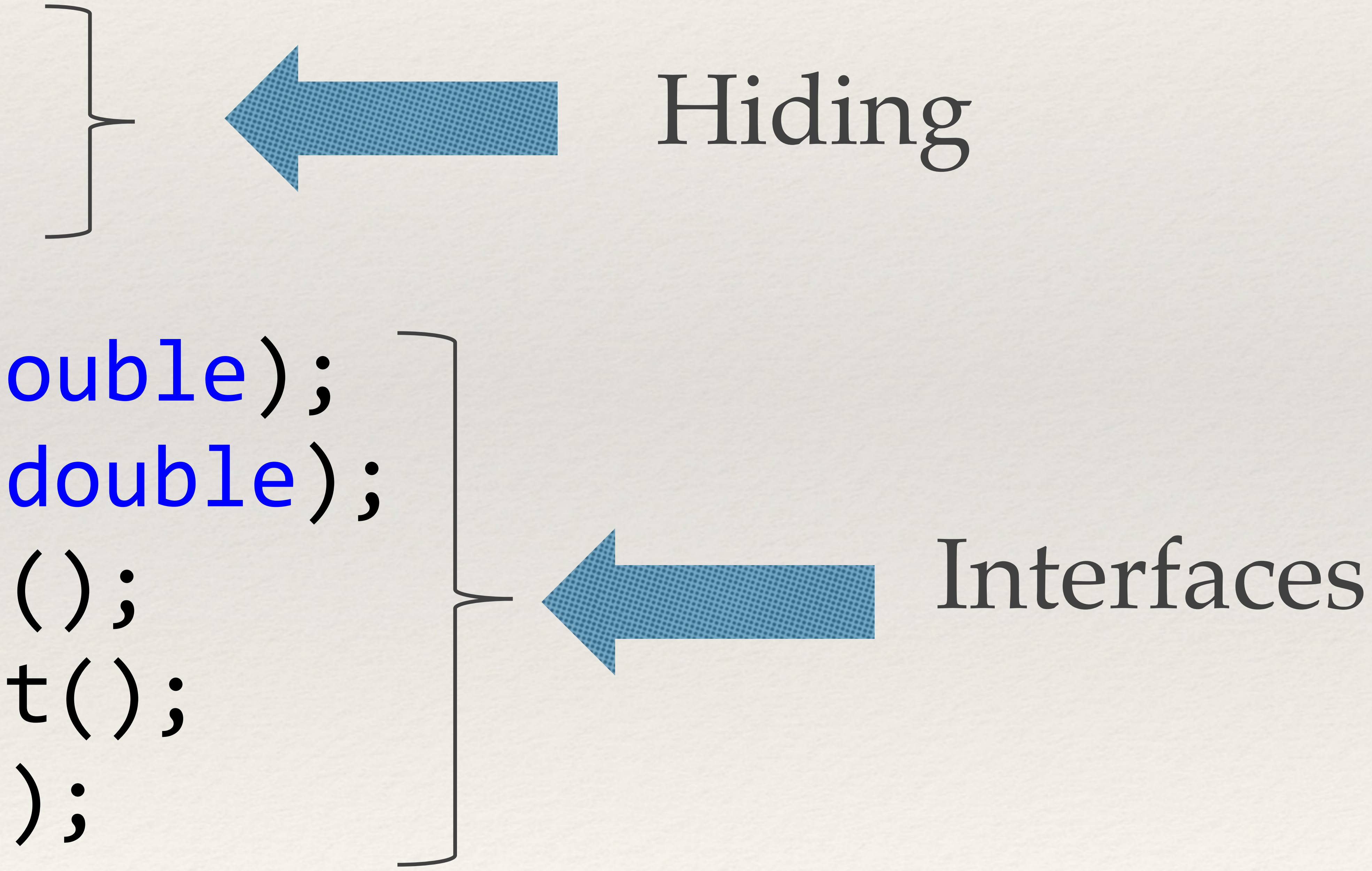
Encapsulation

- Packaging related stuff together
- User need to know only public methods/data of the object: interface
- Interfaces abstract away the details of how all the operations are performed
  - "Data hiding", "black box".



Class Example

```
class Rectangle
{
private:
   double width;
   double height;
public:
   void setWidth(double);
   void setHeight(double);
   double getWidth();
   double getHeight();
   double getArea();
};
```



Hiding

Interfaces

Class Declaration

```
class <Class_Name>
{
<access_specifier>:
   member declaration;
   ...
<access_specifier>:
   member declaration;
   ...
};
```

```
class Rectangle
{
private:
   double width;
   double height;
public:
   void setWidth(double);
   void setHeight(double);
   double getWidth();
   double getHeight();
   double getArea();
};
```

Class Access specifier

- Used to control access to members of the class:
  - private (**default**) : accessible **only** within the class.
  - protected: accessible within the class and its derived classes
  - public: accessible from anywhere outside the class but within the program.
- Can be listed in any order in a class
- Can appear multiple times in a class

Method Member Definition

- When defining a method member function:
    - Put prototype in class declaration
    - Define function using class name and scope resolution operator (::) outside the class

```
void Rectangle::setWidth(double w)
{
   width = w;
}
```

- Or declare a method member function inside the class as a normal function

Declaration vs Definition

- Separate the declaration (specification) part from the definition (implementation) part.
- Place class declaration in a header file, called class specification file. E.g. Rectangle.h
- Place member function definitions in *.cpp file. E.g. Rectangle.cpp. This file must #include the class specification file.
- Programs that use the class must #include the class specification file.

Example

Rectangle.h

```
class Rectangle
{
private:
   double width;
   double height;
public:
   void setWidth(double);
   void setHeight(double);
   double getWidth();
   double getHeight();
   double getArea();
};
```

Rectangle.cpp

```
#include “Rectangle.h”
void Rectangle::setWidth(double w)
{
   width = w;
}
double Rectangle::getWidth()
{
   return width;
}
…
```

Set and Get

- Set (mutator): a member function that stores a value in a private member variable, or changes its value in some way.

```
void setWidth(double);
void setHeight(double);
```

- Get (accessor): a member function that retrieves a value from a private member variable.

```
double getWidth();
double getHeight();
```

Using `const` With Member Functions

- `const` appearing after the parentheses in a member function declaration specifies that the function will not change any data in the calling object.
- Example

```
double getWidth() const;

double getHeight() const;

double getArea() const;
```

Scope operator

- Scope operator `::`
  - Is used in the definition of member function outside the class
  - Inline function vs. normal function
    - Member functions defined in the class definition is considered as inline function.

Static and Instance Members

- Static members: (prefixed by keyword **static**) shared among all objects of the same class.
  - Static field members:
    - Need to be initialized somewhere outside the class
    - Can be accessed through object or class
    - Example: object counter
  - Static method members: can only access static members of the class.
- Instance members: used just for an object.

Access Instance Members

- Must through an object:

  <an object> . <instance member>

  <pointer to an object> -> <instance member>

For example,

```
Rectangle x,*y;
x.getHeight();
y = &x;
y -> getHeight();
```

Access Static Members

- Through an object: like instance members
- Through class name: using scope operator

```
Rectangle::numObject;
```

Constructor vs Destructor

Constructor

- Constructors: a special method that is automatically called whenever a new object is created .
  - allow the class to initialize member variables or allocate storage.
  - no return statement.
  - can not be called explicitly as member methods.

Default Constructor

- A default constructor is a constructor that takes no arguments.
- If you write a class with no constructor at all, C++ will write a default constructor for you, one that does nothing.
- A simple instantiation of a class (with no arguments) calls the default constructor:

```
Rectangle r;
```

Constructor Syntax

```
class <Class_Name>
{
    ...
public:
    <Class_Name>([<list of parameter>]);
    ...
};
```

Constructors with Parameters

- To create a constructor that takes arguments:
  - Indicate parameters in prototype:

    ```
    Rectangle(double , double );
    ```

  - Use parameters in the definition:

    ```
    Rectangle::Rectangle(double w, double h)
    {
       width = w;
       height = h;
    }
    ```

- You can pass arguments to the constructor when you create an object:

  ```
  Rectangle r2(6, 4);
  ```

More About Default Constructors

- If all of a constructor's parameters have default arguments, then it is a default constructor. For example:

```
Rectangle(double = 0, double = 0);
```

- Creating an object and passing no arguments will cause this constructor to execute:

```
Rectangle r;
```

Overloading Constructors

- A class can have more than one constructor. They can be overloaded.
- The compiler automatically call the one whose parameters match the arguments.

```
Rectangle();

Rectangle(double);

Rectangle(double, double);
```

Create an object

- When a variable whose type is a class is declared

```
Rectangle x;
```

- When a new is used

```
Rectangle *x = new Rectangle(2,3);
```

- When an object is assigned

```
Rectangle y = x;
```

- When an object is passed by value

Destructor

- Destructor:
  - responsible for the necessary cleanup of a class when lifetime of an object ends.
  - automatically called when an object is killed
- Destructors have no:
  - return statement
  - parameters
- Destructors must have the same name as the class but prefixed by ~
- Only one destructor per class, i.e., it cannot be overloaded
- If constructor allocates dynamic memory, destructor should release it

Destructor Syntax

```
class <Class_Name>
{
     ...
public:
     ~<Class_Name>();
     ...
};
```

Kill an object

- When a variable keeping the object goes out of scope
- When a dynamically allocated object killed by a delete or delete []

Other Issues

Using Private Member Methods

- A `private` member method can only be called by another member method
- It is used for internal processing by the class, not for use outside of the class
- If you wrote a class that had a public sort function and needed a function to swap two elements, you'd make that private

Arrays of Objects

- Objects can be the elements of an array:

```
Rectangle rooms[8];
```

- Default constructor for object is used when array is defined

Arrays of Objects

- Must use initializer list to invoke constructor that takes arguments:

```
Rectangle rectArray[3]={Rectangle(2.1,3.2),
                       Rectangle(4.1, 9.9),
                       Rectangle(11.2, 31.4)};
```

Accessing Objects in an Array

- Objects in an array are referenced using subscripts

- Member functions are referenced using dot notation:

```
rectArray[1].setWidth(11.3);
cout << rectrArray[1].getArea();
```

Pointer to Class

- Objects can also be pointed by pointers. Class is a valid type.
- Class pointers is similar to struct pointers.
- E.g.:

```
Rectangle r2(6, 4);
Rectangle* r3 = &r2;
cout << r3->getArea() << endl;
cout << (*r3).getArea() << endl;
```

Using the this Pointer

- Every object has access to its own address through a pointer called `this` (a C++ keyword)

```
void Rectangle::setWidth(double width)
{
   this->width = width;
}
```

Summarise

- Understand Class: concept and definition, encapsulation
- Member functions, static and const members
- Constructor/Destructor and overloaded operators