

Object-oriented Programming in C++

Fundamental Computer Programming- C++ Lab (II)



元智大學 資訊工程學系

Department of Computer Science & Engineering

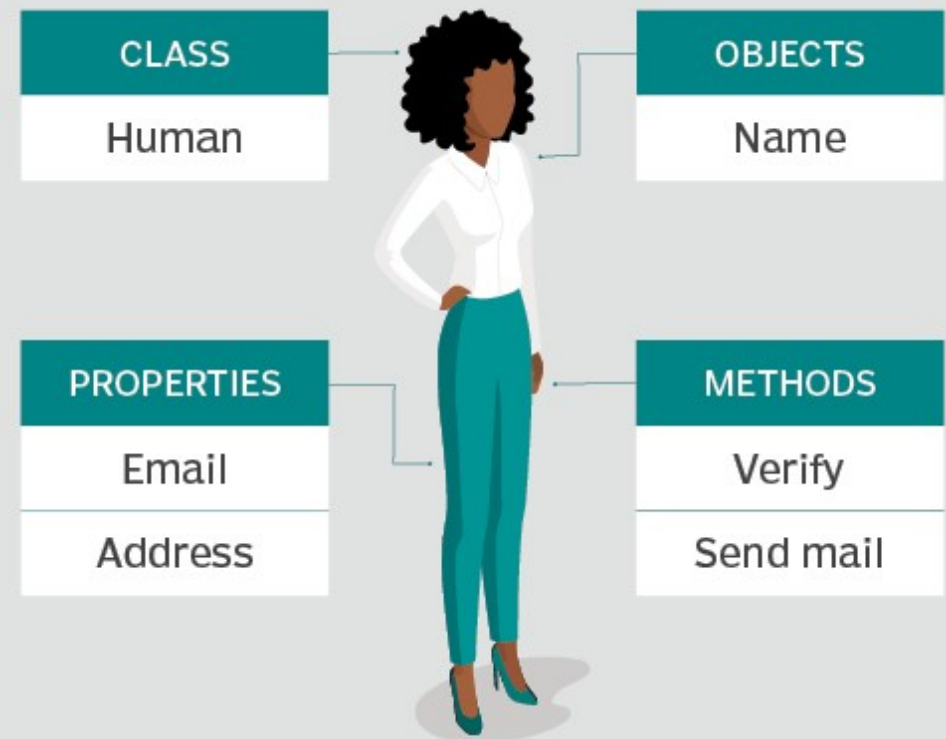
Lecturer: Ho Quang Thai

Outline

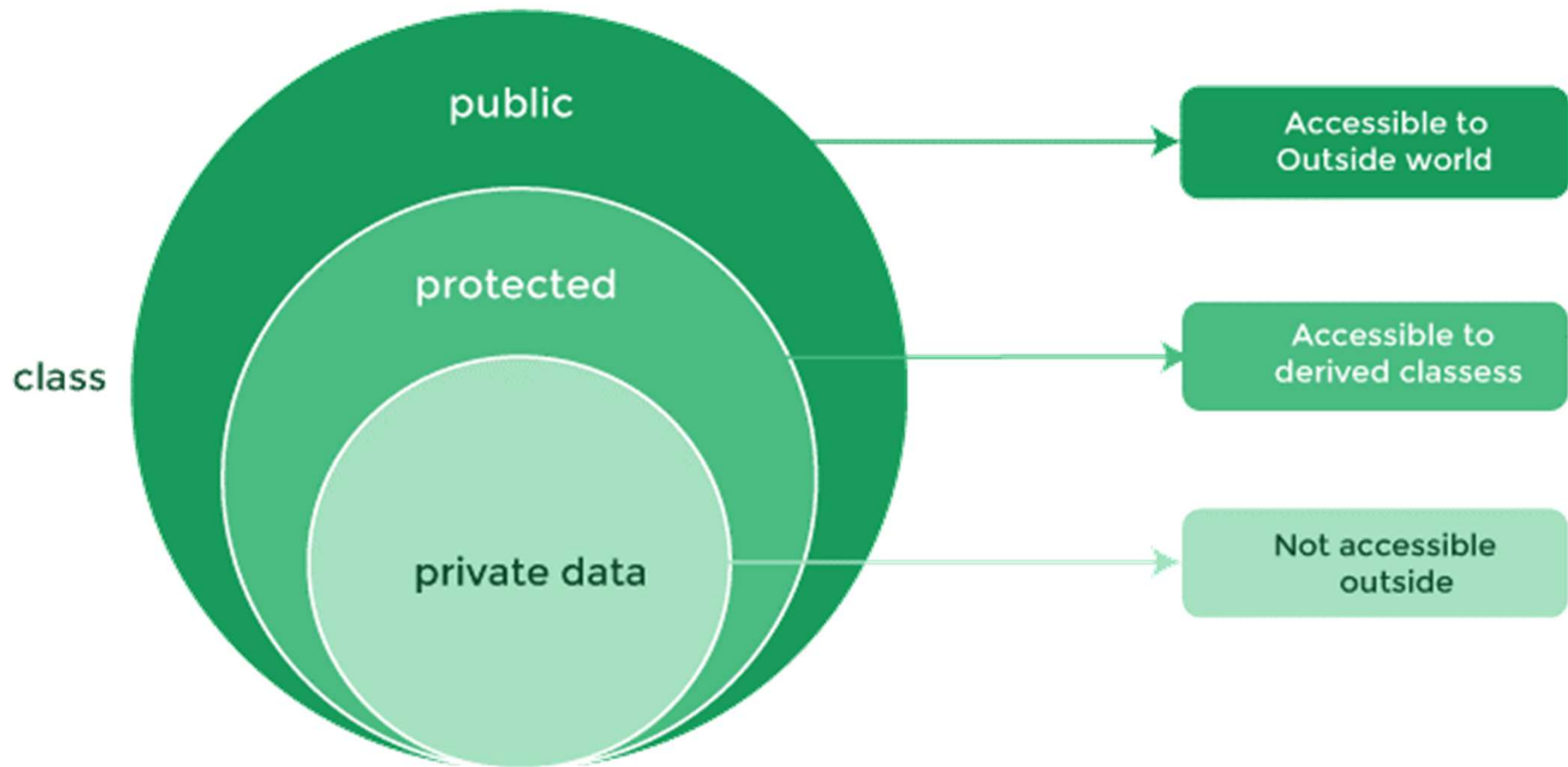
- Object-Oriented Design
- Classes and Object in C++
- Exercises

Object-Oriented Programming

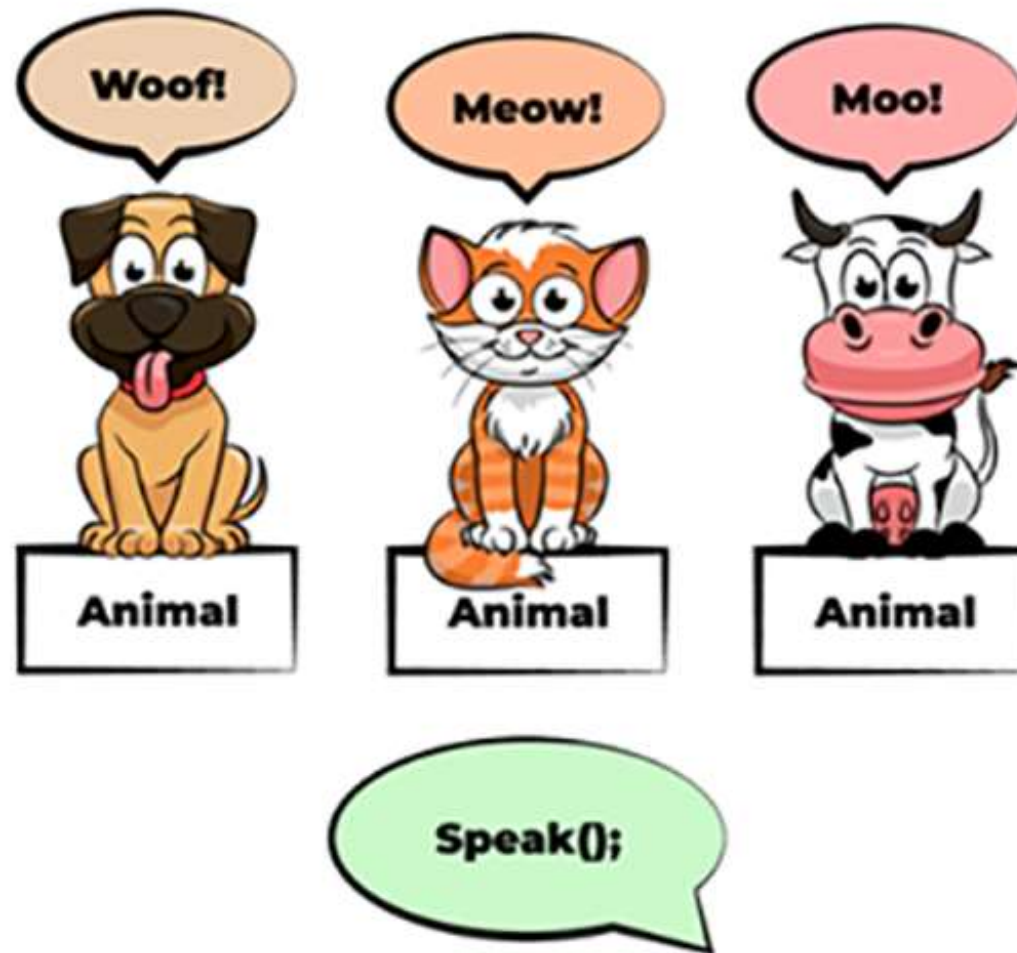
- Break the program down into **objects** and **relationships**, each with **properties** (*data*) and **actions** (*methods*).
- Helps the program easier develop and maintain.
- Concept of OOPs:
 - Encapsulation
 - Inheritance
 - Polymorphism
 - Abstraction



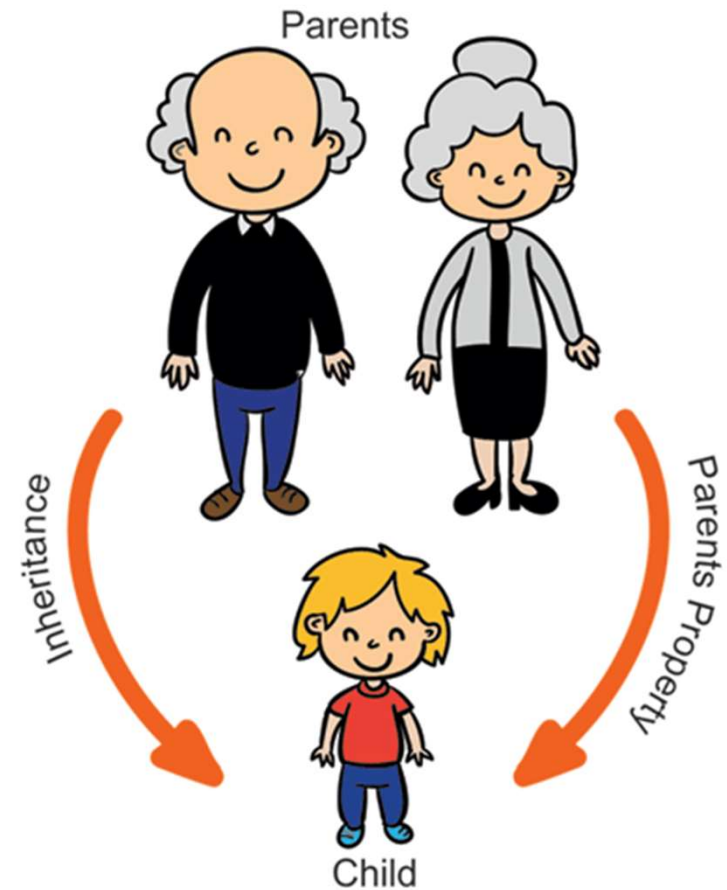
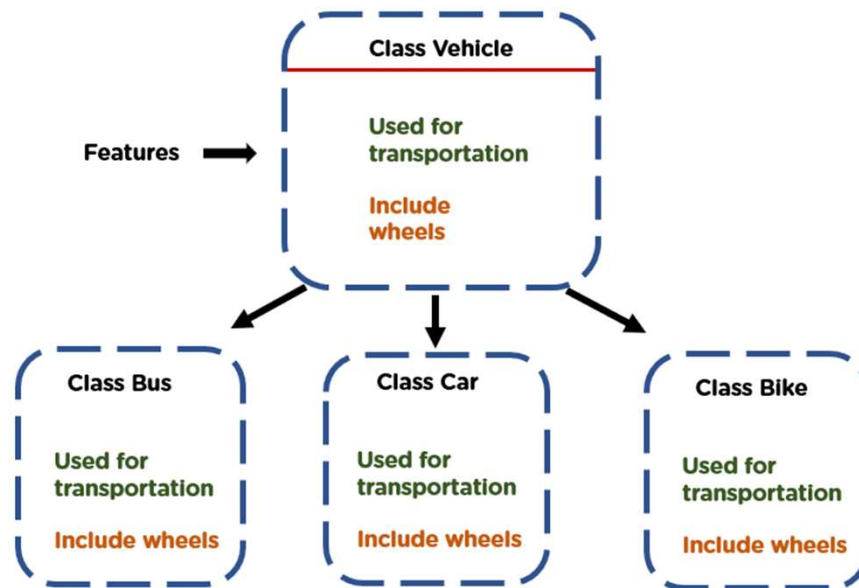
Encapsulation



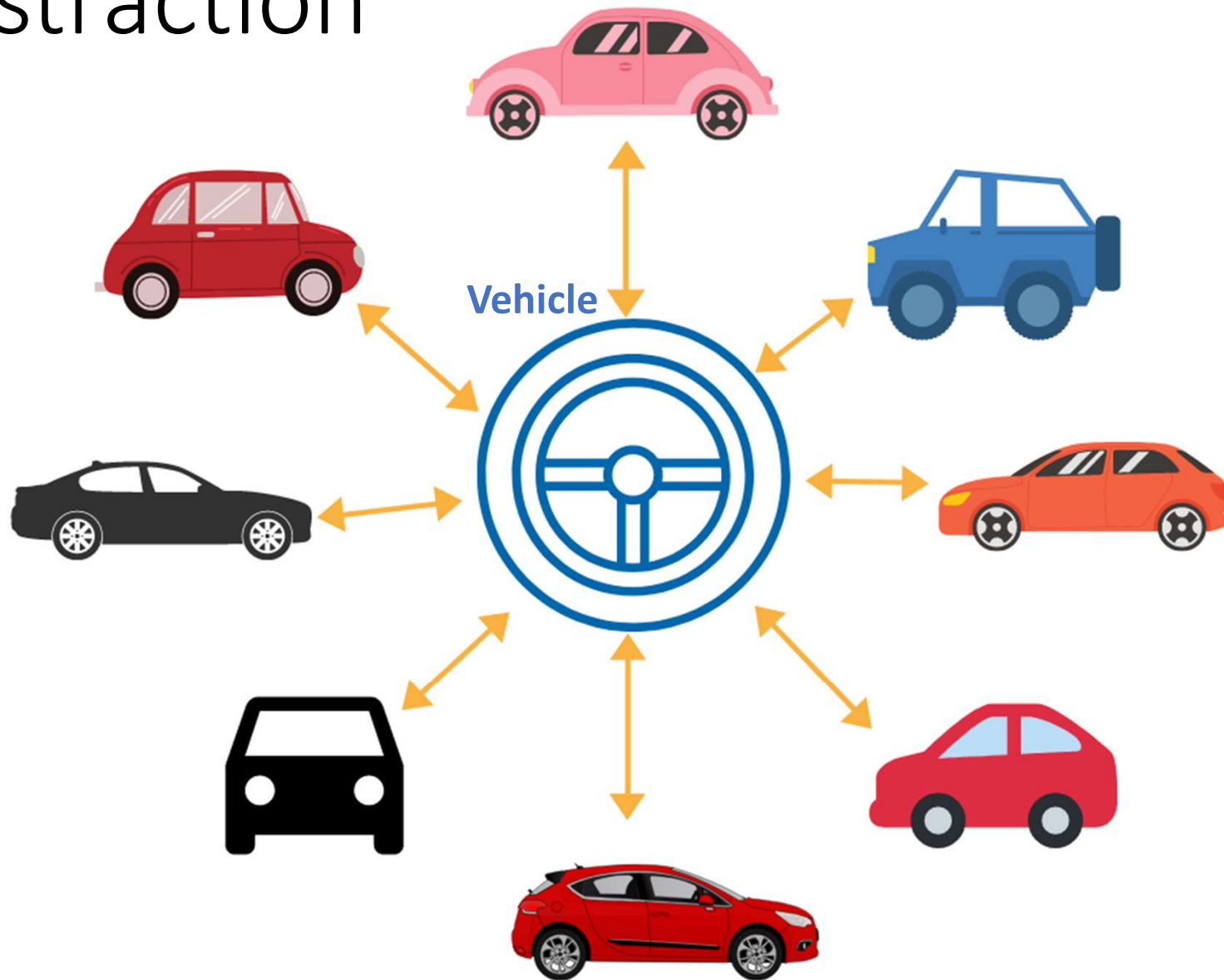
Polymorphism



Inheritance



Abstraction



Classes and Object in C++

- Define Class
- Declare Class Object
- Constructor & Destructor
- Copy Constructor
- Friend Functions
- Keyword `this`
- Pointer to C++ Classes
- Static Member of a Class

Define Class in C++

- A class in C++ is defined using the keyword `class` followed by class name.

```
class Student
{
    public: // Access modifier
           // Data members
           int ID;
           string Name;
           int Age;
           // Member functions
           void getDetails()
           {
               cout << "Student Details\nID:" << ID << endl;
               cout << "Name :" << Name << endl;
               cout << "Age :" << Age << endl;
           }
};
```

Define Class in C++

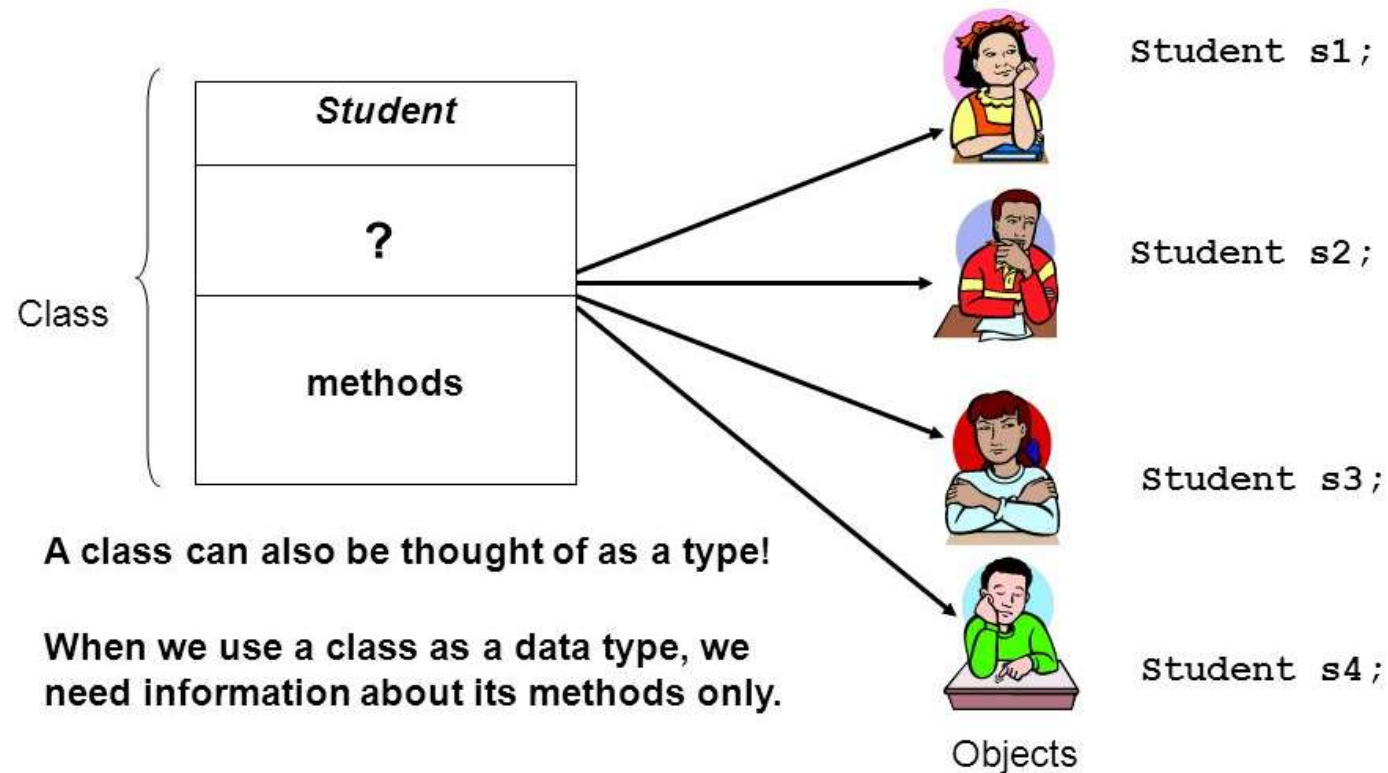
- Example of member function definition outside the class

```
class Student
{
    public:
        int RollNo;
        string Name;
        int Age;
        void getDetails();
};

void Student::getDetails() {
    cout<< "Student Details\nRoll No.:" << RollNo << endl;
    cout<< "Name:" << Name << endl;
    cout<< "Age:" << Age << endl;
}
```

Objects in C++

A **class** is the blueprint from which **objects** are generated;



Objects in C++

- An **instance** of a class is called an object. When a class is defined, no memory is allocated, we only define the specifications for its object. Memory is allocated when we create an object of a class.
- Data members and member functions of a class can be used and accessed by creating objects.
- We can create multiple objects of a class.

```
ClassName ObjectName; // Syntax  
Student s1, s2, s3; // Instances of Student
```

Objects in C++

- An **instance** of a class is called an object.
- Data members and member functions of a class can be used and accessed by creating objects.
- We can create multiple objects of a class.

```
ClassName ObjectName; // Syntax
Student s1, s2, s3; // Instances of Student

Student.Name; //Accessing data member
Student.getDetails(); //Accessing member function
```

Access Modifiers in C++

There are three types of access specifiers in C++:

- **Public:**

- Accessible from anywhere in the program.

- **Private:**

- Only accessible by functions defined within the class.
- Cannot access from outside class except **friend** functions and **friend** classes.

- **Protected:**

- Similar to private, except these can also be accessed by derived classes.

Access Modifiers in C++

Example

Name:Vidvan
Age:10yrs

```
#include <iostream>
#include <string>

using namespace std;

class Person {
    private:
        string name;
        int age;
    public:
        void insert(string n, int a) {
            name = n;
            age = a;
        }
        void display() {
            cout<<"Name:"<<name<<endl;
            cout<<"Age:"<<age<<"yrs";
        }
};

int main() {
    Person p1;
    p1.insert("Vidvan", 10);
    p1.display();
    return 0;
}
```

Constructor

- A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will **have exact same name as the class** and it does not have any return type at all, not even void.
- Constructors can be very useful for setting initial values for certain member variables.

Constructor

Example

Object is being created
Length of line : 6

```
#include <iostream>

using namespace std;

class Line {
public:
    void setLength( double len );
    double getLength( void );
    Line(); // This is the constructor
private:
    double length;
};

// Member functions definitions including constructor
Line::Line(void) {
    cout << "Object is being created" << endl;
}

void Line::setLength( double len ) {
    length = len;
}

double Line::getLength( void ) {
    return length;
}

// Main function for the program
int main() {
    Line line;

    // set line length
    line.setLength(6.0);
    cout << "Length of line : " << line.getLength() << endl;

    return 0;
}
```

Parameterized Constructor

Example

Object is being created,
length = 10
Length of line : 10
Length of line : 6

```
#include <iostream>

using namespace std;
class Line {
public:
    void setLength( double len );
    double getLength( void );
    Line(double len); // This is the constructor

private:
    double length;
};

// Member functions definitions including constructor
Line::Line( double len) {
    cout << "Object is being created, length = " << len << endl;
    length = len;
}

void Line::setLength( double len ) {
    length = len;
}

double Line::getLength( void ) {
    return length;
}

// Main function for the program
int main() {
    Line line(10.0);

    // get initially set length.
    cout << "Length of line : " << line.getLength() << endl;

    // set line length again
    line.setLength(6.0);
    cout << "Length of line : " << line.getLength() << endl;

    return 0;
}
```

Using Initialization Lists to Initialize Fields

In case of parameterized constructor, you can use following syntax to initialize the fields

```
Line::Line( double len): length(len) {  
    cout << "Object is being created, length = " << len << endl;  
}
```

Above syntax is equal to the following syntax –

```
Line::Line( double len) {  
    cout << "Object is being created, length = " << len << endl;  
    length = len;  
}
```

If for a class C, you have multiple fields X, Y, Z, etc., to be initialized, then use can use same syntax and separate the fields by comma as follows –

```
C::C( double a, double b, double c): X(a), Y(b), Z(c) {  
    ....  
}
```

Destructor

- A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will **have exact same name as the class** prefixed with **a tilde (~)** and it can neither return a value nor can it take any parameters.
- Destructor can be very useful for **releasing resources** before coming out of the program like closing files, releasing memories etc.

Destructor

Example

Object is being created
Length of line : 6
Object is being deleted

```
#include <iostream>

using namespace std;
class Line {
public:
    void setLength( double len );
    double getLength( void );
    Line();    // This is the constructor declaration
    ~Line();   // This is the destructor: declaration

private:
    double length;
};

// Member functions definitions including constructor
Line::Line(void) {
    cout << "Object is being created" << endl;
}
Line::~~Line(void) {
    cout << "Object is being deleted" << endl;
}
void Line::setLength( double len ) {
    length = len;
}
double Line::getLength( void ) {
    return length;
}

// Main function for the program
int main() {
    Line line;

    // set line length
    line.setLength(6.0);
    cout << "Length of line : " << line.getLength() << endl;

    return 0;
}
```

Copy Constructor

- The **copy constructor** is a constructor which creates an object by initializing it with an object of the same class
- The copy constructor is used to:
 - Initialize one object from another of the same type.
 - Copy an object to pass it as an argument to a function.
 - Copy an object to return it from a function.
- If a copy constructor is not defined in a class, the *compiler itself defines one*.
- If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor.
- The most common form of copy constructor is shown here –

```
ClassName (const ClassName &obj) { // Constructor  
|   // body of constructor  
}
```

Copy Constructor

Example

p1.x = 10, p1.y = 15
p2.x = 10, p2.y = 15

```
#include <iostream>
using namespace std;

class Point {
private:
    int x, y;

public:
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    // Copy constructor
    Point(const Point& p1)
    {
        x = p1.x;
        y = p1.y;
    }

    int getX() { return x; }
    int getY() { return y; }
};

int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX()
         << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX()
         << ", p2.y = " << p2.getY();
    return 0;
}
```


Friend Functions

- A **friend function** of a class is defined outside a class, but it has the right to **access all private and protected members of the class**.
- A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all its members are friends.

Width of box : 10

```
#include <iostream>

using namespace std;

class Box {
    double width;

    public:
        friend void printWidth( Box box );
        void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid ) {
    width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box ) {
    /* Because printWidth() is a friend of Box, it can
       directly access any member of this class */
    cout << "Width of box : " << box.width << endl;
}

// Main function for the program
int main() {
    Box box;

    // set box width without member function
    box.setWidth(10.0);

    // Use friend function to print the width.
    printWidth( box );

    return 0;
}
```


Keyword `this`

- `this` pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.
- Friend functions do not have a `this` pointer, because friends are not members of a class. Only member functions have a `this` pointer.

Constructor called.
Constructor called.
Box2 is equal to or larger
than Box1

```
#include <iostream>

using namespace std;

class Box {
public:
    // Constructor definition
    Box(double l = 2.0, double b = 2.0, double h = 2.0) {
        cout << "Constructor called." << endl;
        length = l;
        breadth = b;
        height = h;
    }
    double Volume() {
        return length * breadth * height;
    }
    int compare(Box box) {
        return this->Volume() > box.Volume();
    }

private:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box
};

int main(void) {
    Box Box1(3.3, 1.2, 1.5);    // Declare box1
    Box Box2(8.5, 6.0, 2.0);    // Declare box2

    if(Box1.compare(Box2)) {
        cout << "Box2 is smaller than Box1" << endl;
    } else {
        cout << "Box2 is equal to or larger than Box1" << endl;
    }

    return 0;
}
```

Pointer to C++ Classes

- A pointer to a C++ class is done the same way as a pointer to a structure and to access members of a pointer to a class you use the member access operator **->** operator.

```
Constructor called.  
Constructor called.  
Volume of Box1: 5.94  
Volume of Box2: 102
```

```
#include <iostream>

using namespace std;

class Box {
public:
    // Constructor definition
    Box(double l = 2.0, double b = 2.0, double h = 2.0) {
        cout << "Constructor called." << endl;
        length = l;
        breadth = b;
        height = h;
    }
    double Volume() {
        return length * breadth * height;
    }

private:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box
};

int main(void) {
    Box Box1(3.3, 1.2, 1.5);    // Declare box1
    Box Box2(8.5, 6.0, 2.0);    // Declare box2
    Box *ptrBox;                // Declare pointer to a class.

    // Save the address of first object
    ptrBox = &Box1;

    // Now try to access a member using member access operator
    cout << "Volume of Box1: " << ptrBox->Volume() << endl;

    // Save the address of second object
    ptrBox = &Box2;

    // Now try to access a member using member access operator
    cout << "Volume of Box2: " << ptrBox->Volume() << endl;

    return 0;
}
```

Static member

- A pointer to a C++ class is done the same way as a pointer to a structure and to access members of a pointer to a class you use the member access operator **->** operator.

Initial Stage Count: 0
Constructor called.
Constructor called.
Final Stage Count: 2

```
#include <iostream>

using namespace std;

class Box {
public:
    static int objectCount;

    // Constructor definition
    Box(double l = 2.0, double b = 2.0, double h = 2.0) {
        cout << "Constructor called." << endl;
        length = l;
        breadth = b;
        height = h;

        // Increase every time object is created
        objectCount++;
    }

    double Volume() {
        return length * breadth * height;
    }

    static int getCount() {
        return objectCount;
    }

private:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box
};

// Initialize static member of class Box
int Box::objectCount = 0;

int main(void) {
    // Print total number of objects before creating object.
    cout << "Initial Stage Count: " << Box::getCount() << endl;

    Box Box1(3.3, 1.2, 1.5);    // Declare box1
    Box Box2(8.5, 6.0, 2.0);    // Declare box2

    // Print total number of objects after creating object.
    cout << "Final Stage Count: " << Box::getCount() << endl;

    return 0;
}
```

Exercise 1

Write a **Student** class as design.

- **Input()**: Enter info of student
- **Ouput()**: Print infor of student to the screen.

Write a **main** function to test the class.

Student
string ID string Name int Age double Score
void Input() void Output()

Exercise 2

Write a **Rectangle** class as design.

- **Input()**: Enter info of the rectangle
- **Draw()**: Output the rectable to the screen by using * symbol
- **Area()**: Calculate and return area of the rectangle.
- **Perimeter()**: Calculate and return perimeter of the rectangle.

Write a **main** function to test the class.

Rectangle
int Width int Length
void Input() void Draw() double Area() double Perimeter()

Formulas

Area of a rectangle = Length × Width

Perimeter of a rectangle = 2(Length + Width)

Exercise 3

Write a **Product** class as design.

- **Input()**: Enter info of that product
- **Output()**: Print info of that product in one line to the screen.
- Write a **main** function to input n of products. Then print all info of all products to the screen.

Product
string ID string Name double Price int Quantity
void Input() void Output()

Exercise 4

Write a **Book** class as design.

- **Input()**: Enter info of that book
- **Output()**: Print info of that book in one line to the screen.
- Write a **main** function to input n of books. Then print all info of all books to the screen.

Book
string Code string Name string Publisher int NumPage double Price
void Input() void Output()

Thank you for your attention