

Introduction to Object Oriented Programming (OOP)

Ali Haider

syedalihaider.ciit@gmail.com

Department of Computer Science IUB

Overview

- Why we need OOP?
- OOP Characteristics
- Classes
- Objects

Why We Need OOP

- Limitations found in earlier approaches
- **Earlier Approaches are**
- Procedural Languages (c, pascal, fortran-List of instructions)
- Functional Programming (list of instruction group in functions)
- Structure Programming (function grouping for creating module)

Real World Modeling

- Objects (attributes+behavior)
- In object-oriented programming, objects are modeled to real-world objects.
- A real-world object has actions related to it and characteristics of its own.
- Take a ball, for example: A ball can be acted on—rolled, tossed, thrown, bounced, caught.
- But it also has its own physical characteristics—size, shape, composition, weight, color, speed, position.
- An accurate data model of a real ball would define not only the physical characteristics but all related actions and characteristics in one package

OOP Approach

- Combines data and functions for formation of object
- **Object-Oriented Programming (OOP)** is the term used to describe a programming approach based on **objects** and **classes**.
- The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behavior.

Characteristics of OOP

- Objects
- Classes
- Inheritance
- Reusability
- Creating new data types
- Polymorphism and operator overloading

Objects

- Physical (air craft in an air traffic control system)
- Elements of Computer-User Environment(windows, menus, graphic objects like line, circle)
- Data Storage Constructs (stacks,linked lists,binary tree)
- Human Entities (Students, Employees)

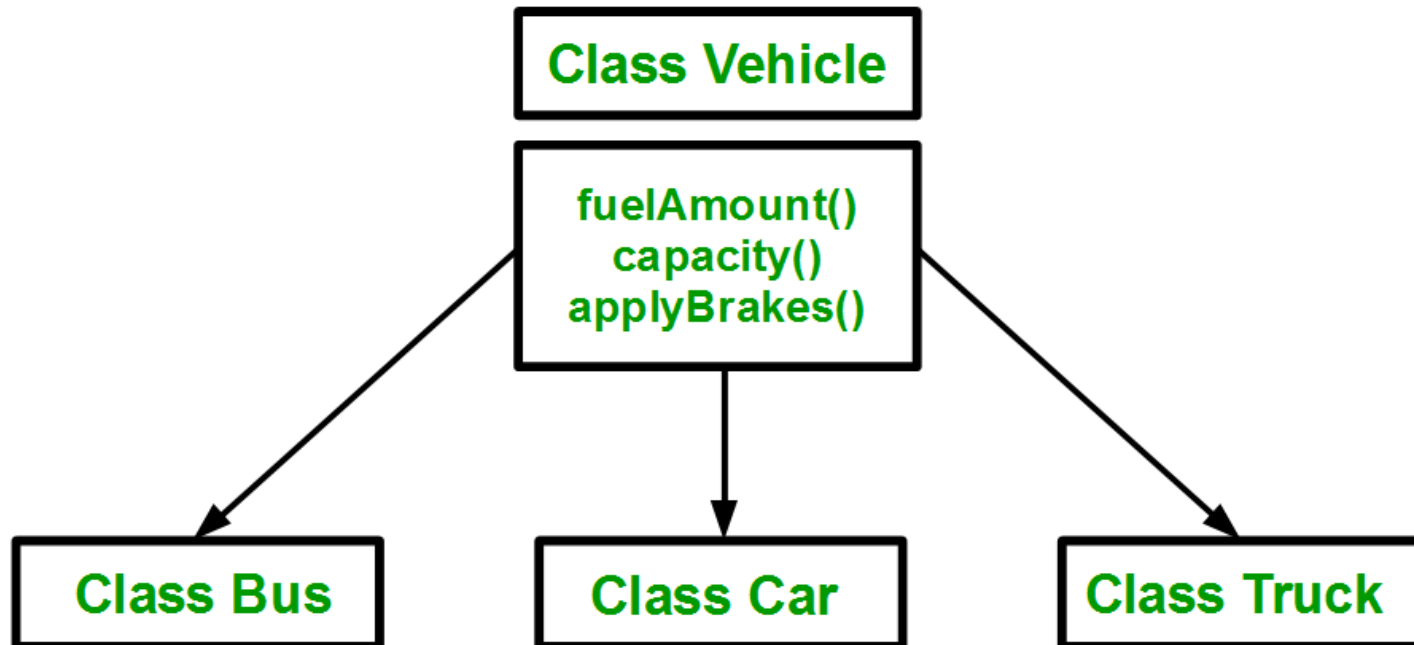
Classes

- A class is blueprint of object (specify the data and function of objects)
- Classes are like templates; they define the methods and variables that a group of similar objects have in common and store them in one place

[illegible]

Inheritance

- Enabling the sharing of common characterizes with subclasses
- Parent child relationship



Reusability

- Created classes can be reused for different projects
- This helps in to use the same code over and again without writing it multiple times which saves time, coding effort and code reduction.

Creating new datatypes

- Using 2D positioning system (x,y) plane
- New datatype will be position
- class position{
- Int x;
- Int y;
- void getData(int a,int b){
- x=a;
- y=b
- }
- void showData(){ //write code for print value of x and y}
- };

Creating Class

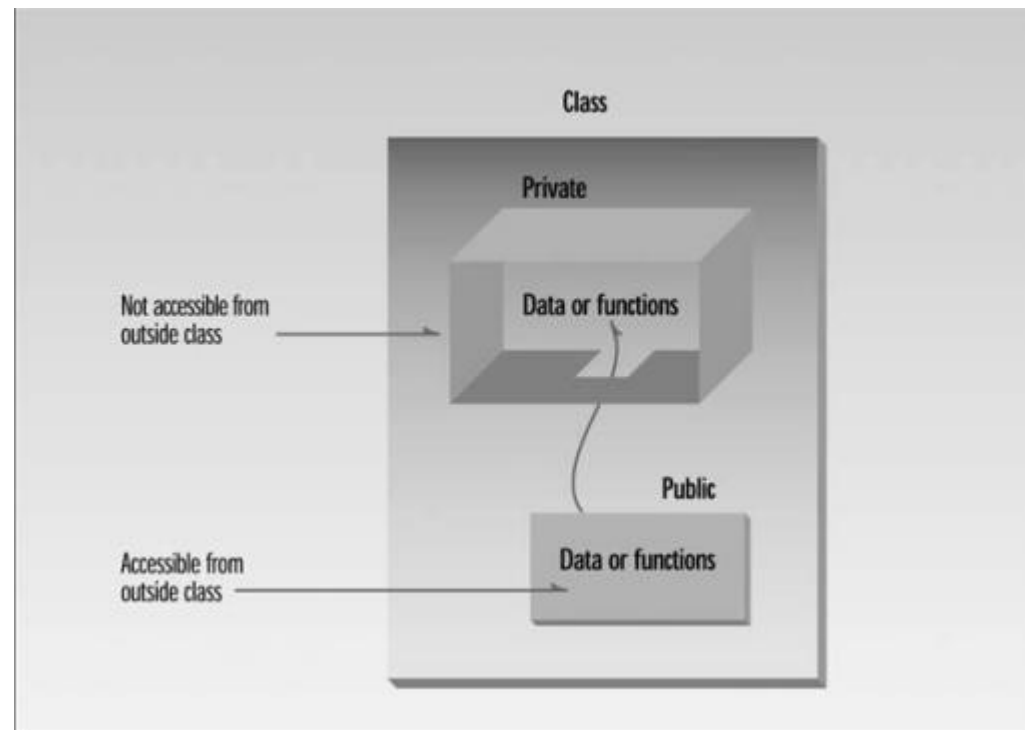
- class student
- {
- private:
- Data Members
- public:
- Member Functions
- };

Example

```
#include<iostream>
#include<string>
using namespace std;
class Car {
    private:
        string brand;
        string model;
        int year;
    public:
        void Get(string b,string m,int y){
            brand=b;
            model=m;
            year=y;
        }
        void Show()
        {
            cout<<"Brand "<<brand<<endl;
            cout<<"Model "<<model<<endl;
            cout<<"Year "<<year<<endl;
        }
};
int main(){
    Car c;
    c.Get("Toyota","Crolla",2002);
    c.Show();
    return 0;
}
```

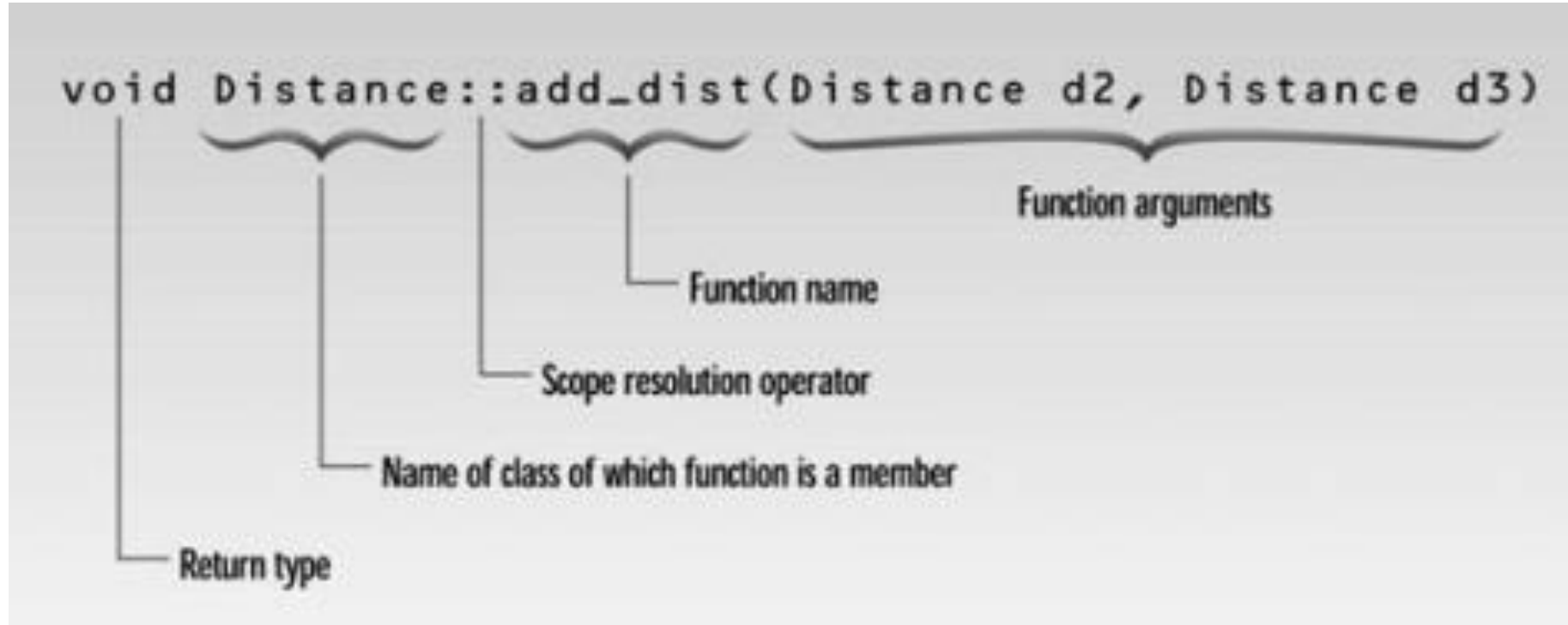
Access Specifiers

- A key feature of object-oriented programming is data hiding.
- Public, Private and Protected



Define Member Function Outside of Class

- Return_Type Class_Name :: Function_Name (Parameters)



Example

```
#include <iostream>
#include <conio.h>
using namespace std;
class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }
    void show(); //function prototype
};
//fuction definition
void Point::show()
{
    cout << "x = " <<x << ", y = " << y<<endl;
}
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
    p1.show();
    p2.show();

    return 0;
}
```