

HI,
I AM SUMAIYAH



Email : Sumaiyah@nu.edu.pk

Office : In Front of CS Secretariat



MARKS DISTRIBUTION

Mid-1	:	15
Mid-2	:	15
Assignment:		08
Quizzes	:	12
Final	:	50

P.S. THESE SLIDES ARE USELESS IF YOU DO
NOT ATTEND CLASSES

NOTE: ALL THE MATERIALS TAKEN FROM
EXTERNAL WEBSITES ARE LINKED IN THE
REFERENCE SECTION

OBJECT ORIENTED PROGRAMMING

PROGRAMMING PARADIGM

paradigm = method to solve a problem

programming paradigm = an overall approach to writing program code

E.g.

Procedural programming

Object Oriented Programming

PROCEDURAL PROGRAMMING

Program code is divided up into procedures.

Discrete blocks of code that carry out a single task.

Also referred as **top-down problem-solving**.

Splitting code into smaller chunks has many benefits:

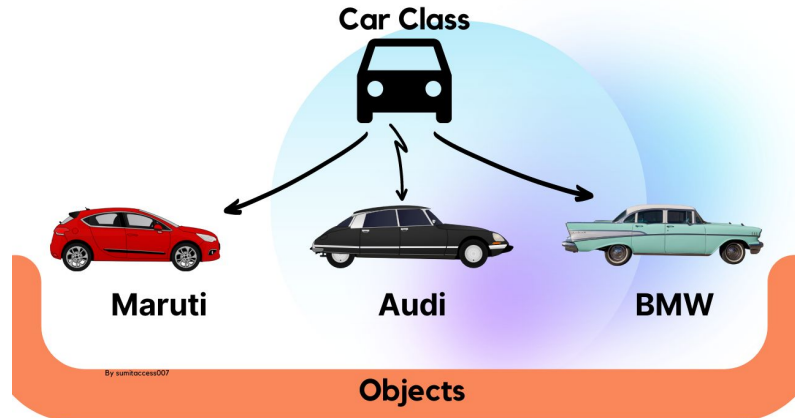
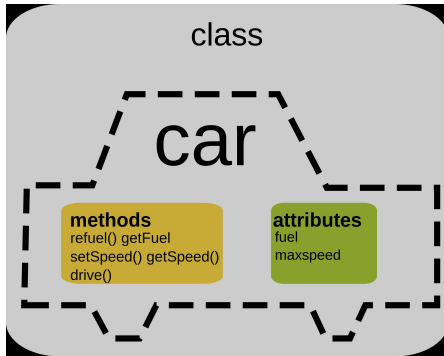
1. It is much easier to test and debug, e.g. tracing 20 lines of code instead of 200 lines of code.
2. Procedures can be called many times, reducing the amount of repeated code.
3. Procedures can manipulate shared data.

OBJECT ORIENTED PROGRAMMING

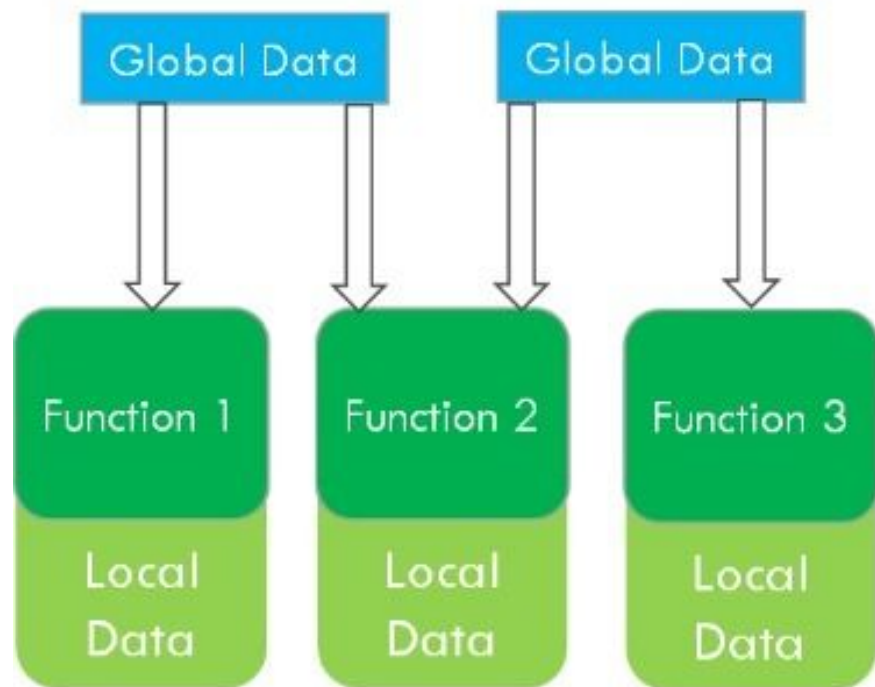
Object = contains data and code

Class = Definition or blueprint of the object

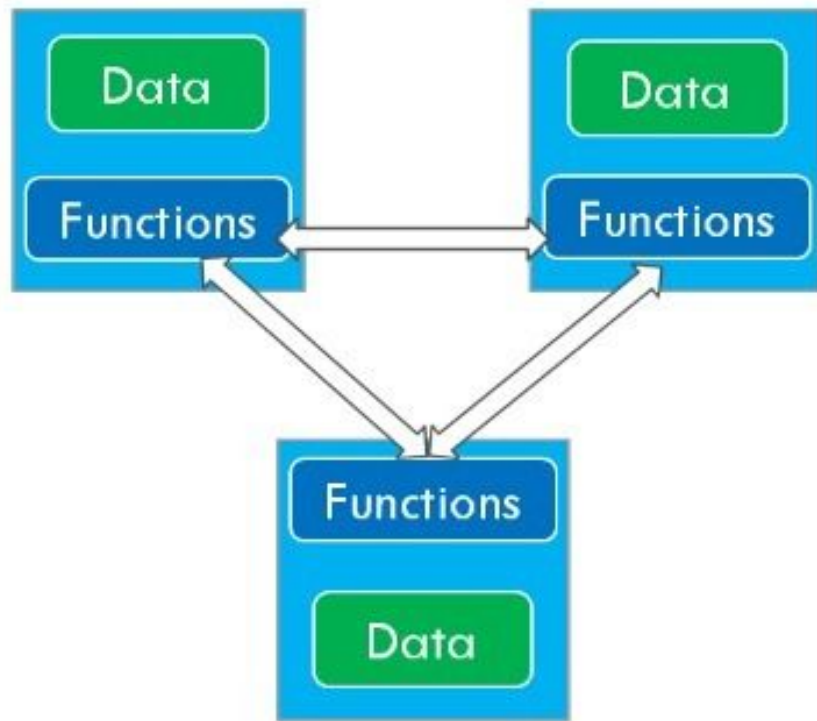
Classes contain data members and member functions.



Procedural Oriented Programming



Object Oriented Programming



PROCEDURAL PROGRAMMING VS OBJECT ORIENTED PROGRAMMING

Procedural paradigm focuses on **what needs to be done**, rather than on the **integrity of the data** that it manipulates.

Global variables can be accessed and modified by every subroutine in a program.

Procedure programming follows **top down** approach.

OOP focuses on the **objects** that make up the system.

Object oriented programming follows **bottom up** approach.

C VS C++

C

Supports procedural programming paradigm

Polymorphism, encapsulation, abstraction, and inheritance are not supported

Access specifiers are not supported. No default mechanism to hide data

Function **overloading** and default parameters are not supported

Namespace is not supported

Virtual function and **friend** function are not supported

C++

Known as **hybrid language** because it supports both procedural and object oriented programming paradigm

Polymorphism, encapsulation, abstraction, and inheritance are supported

Support access specifiers & data hiding

Function **overloading** and default parameters are supported

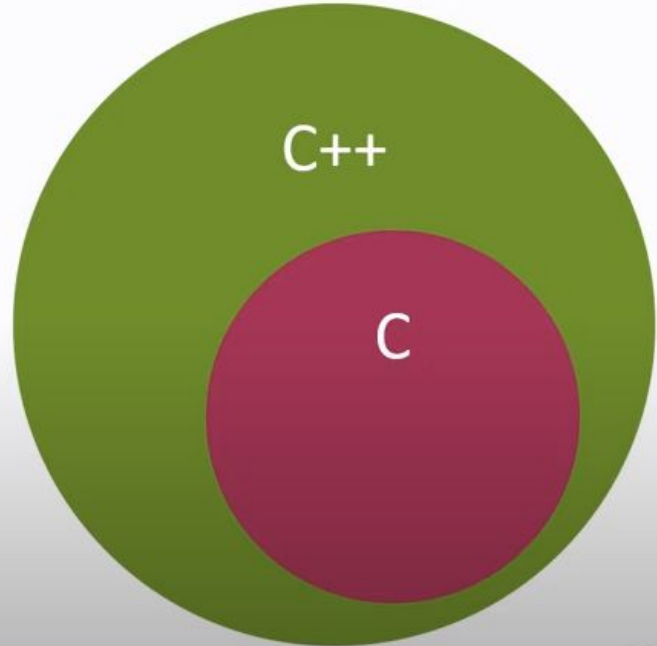
Namespace is supported

Virtual function and **friend** function are supported

C VS C++



C++ can be said a superset of C.



OBJECT ORIENTED MODEL

- Purpose is to understand the product before developing it.
- People think in terms of objects
- OO models map to reality

Therefore, OO models are:

- easy to develop
- easy to understand



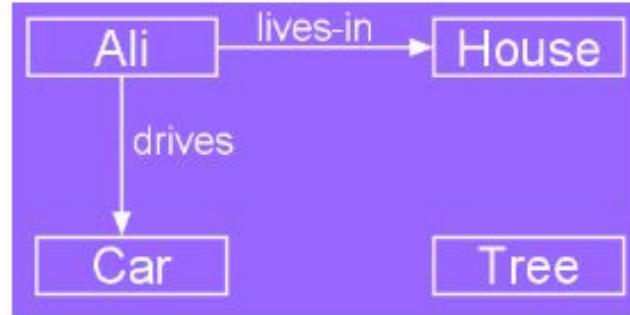
OBJECT ORIENTED MODEL

Objects

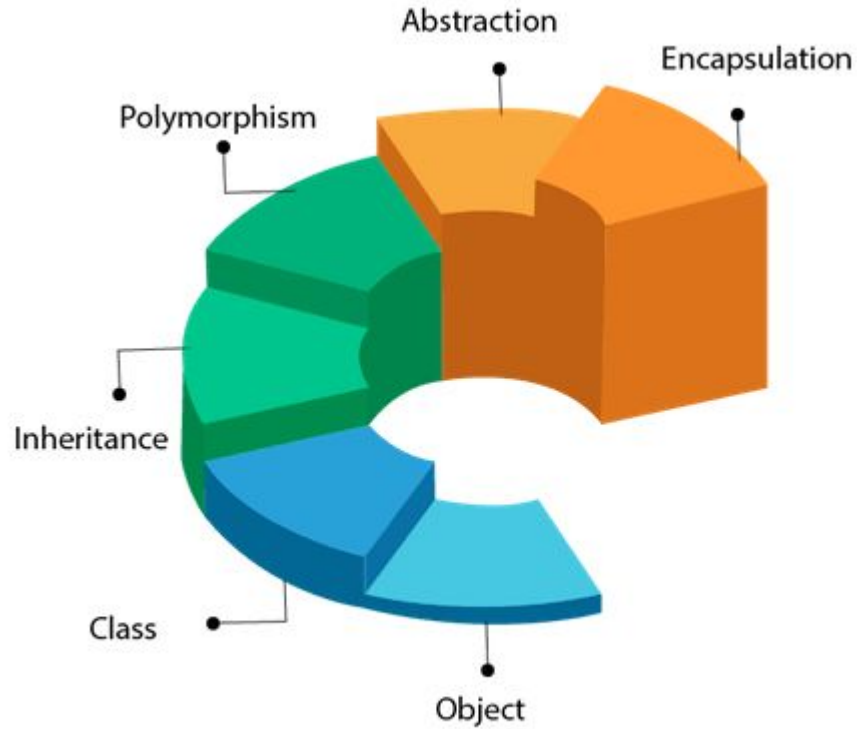
- Person i.e Name: Ali
- House
- Car
- Tree

Interactions

- Ali lives in the house
- Ali drives the car



OBJECT ORIENTED PROGRAMMING SYSTEM



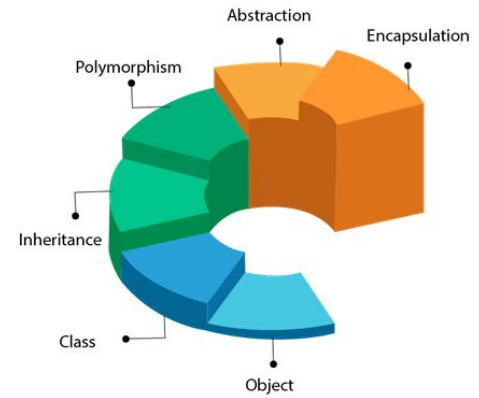
OBJECT ORIENTED PROGRAMMING SYSTEM

Inheritance: Extending one class to another. It provides code reusability.

Polymorphism: Performing one task in different ways.

Abstraction: Hiding internal details and showing functionality.

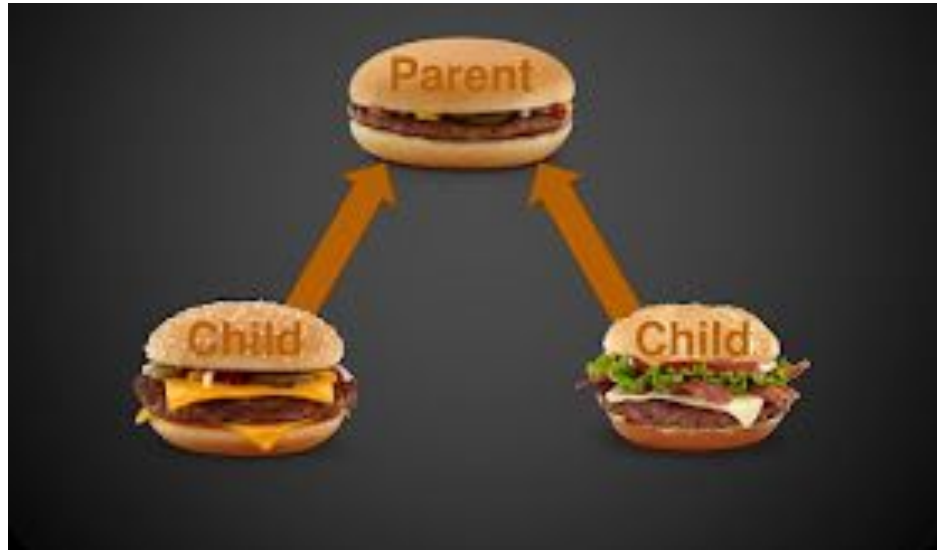
Encapsulation: Binding (or wrapping) code and data together into a single unit.



Capsule

OBJECT ORIENTED PROGRAMMING SYSTEM

Inheritance: Extending one class to another. It provides code reusability.



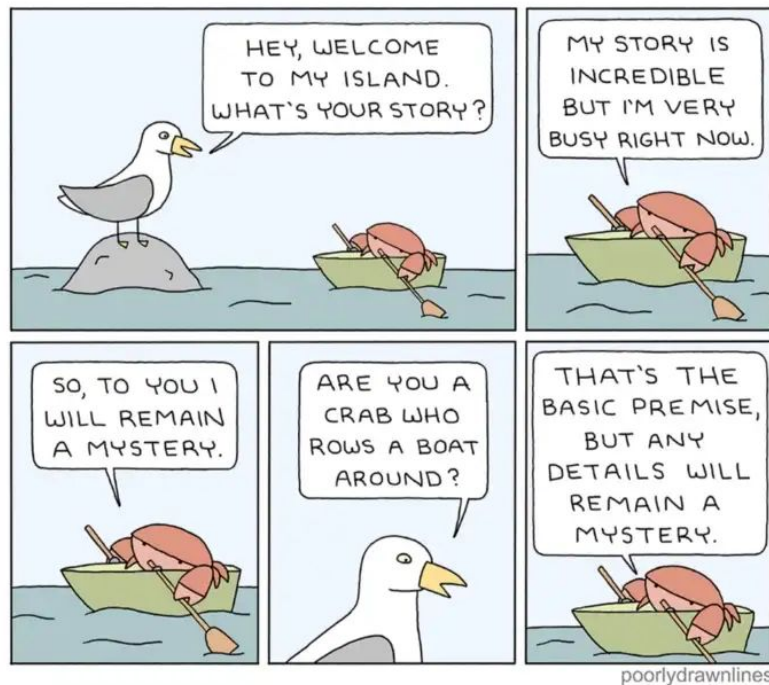
OBJECT ORIENTED PROGRAMMING SYSTEM

Polymorphism: Performing one task in different ways.



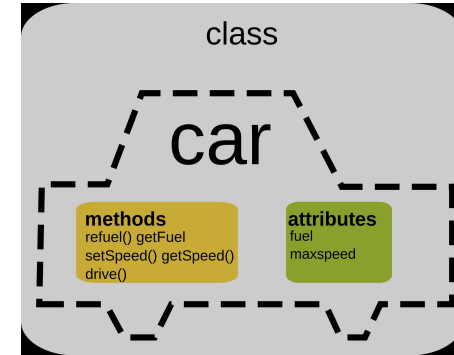
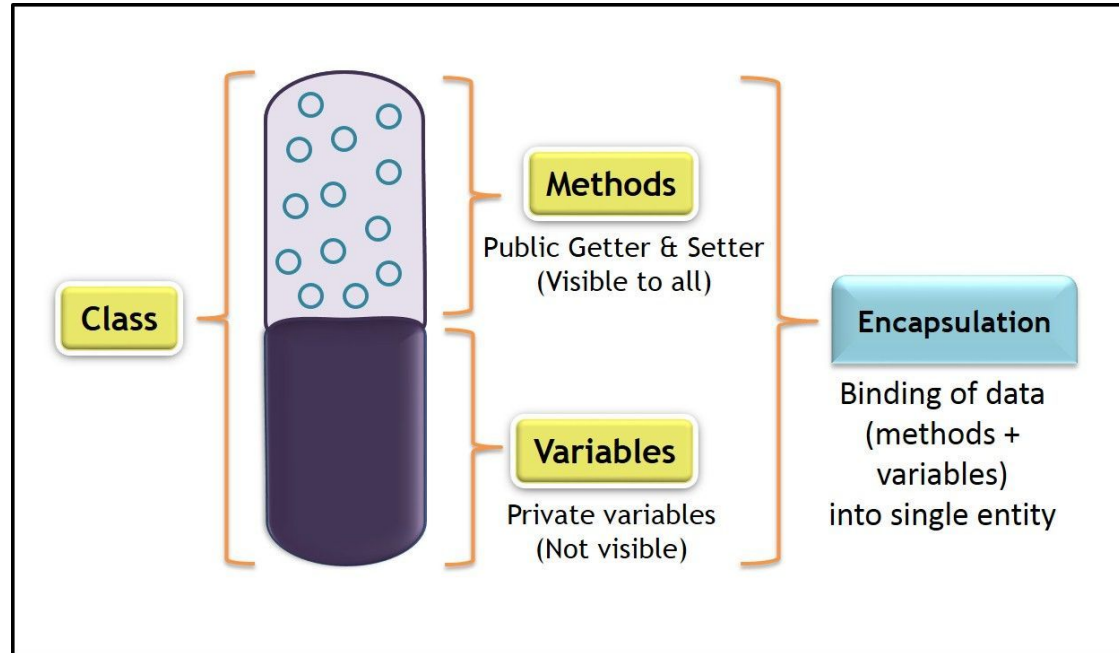
OBJECT ORIENTED PROGRAMMING SYSTEM

Abstraction: Hiding internal details and showing functionality.



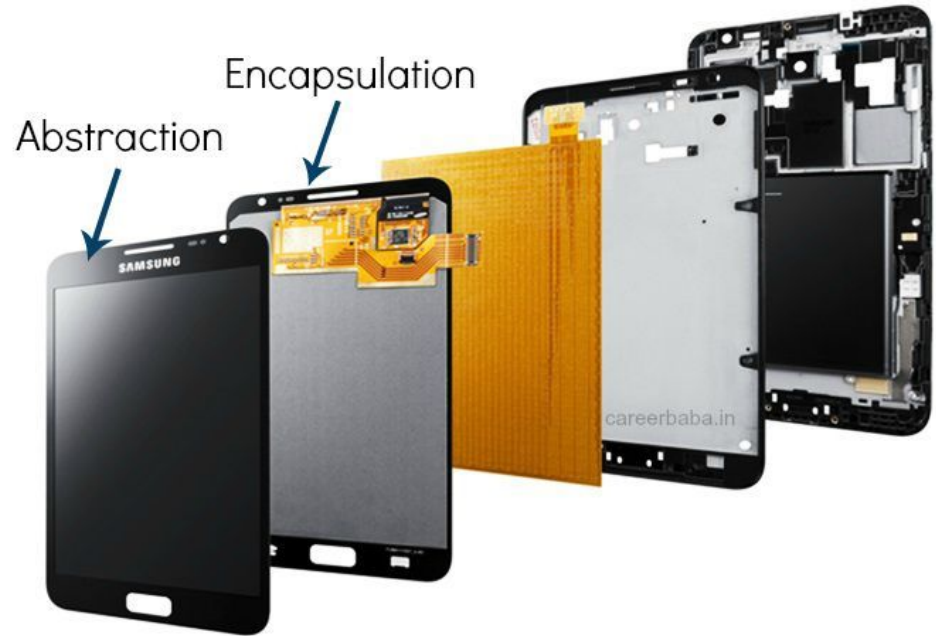
OBJECT ORIENTED PROGRAMMING SYSTEM

Encapsulation: Binding (or wrapping) code and data together into a single unit.



ABSTRACTION AND ENCAPSULATION

Encapsulation leads to
“Data Abstraction”



OOPS Concept

As memes..!

INHERITANCE



ABSTRACTION



POLYMORPHISM



ENCAPSULATION



WHAT IS AN OBJECT?

An object is:

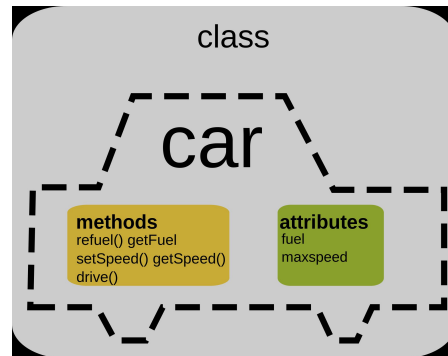
- It can be anything for which we want to save Information
- Something tangible (Ali, Car)
- Something that can be captured intellectually (Time, date)

An object has:

State / attributes / properties / data

Well-defined behavior / methods / functions

Unique identity



ALI AS AN OBJECT?

Attributes:

- Name
- Age

Behavior (operations)

- Walks
- Eats

Identity

- His name

CAR AS AN OBJECT?

Attributes:

- ????

Behavior (operations)

- ????

Identity

- ????

CAT AS AN OBJECT?

Attributes:

- ????

Behavior (operations)

- ????

Identity

- ????

WHAT IS AN OBJECT?

Attributes:

- ????

Behavior (operations)

- ????



Humans



BANK ACCOUNT AS AN OBJECT?

Attributes:

- ????

Behavior (operations)

- ????

Identity

- ????

FLIGHT TICKET AS AN OBJECT?

Attributes:

- ????

Behavior (operations)

- ????

Identity

- ????

WHAT IS A CLASS?

- Collection of Similar object.
- The objects that share some common features.
- It is the a design detail of an object.
- It tell us what an object contains in it.

Technical Definition:

“ A class is blueprint of an object”

IN A NUTSHELL

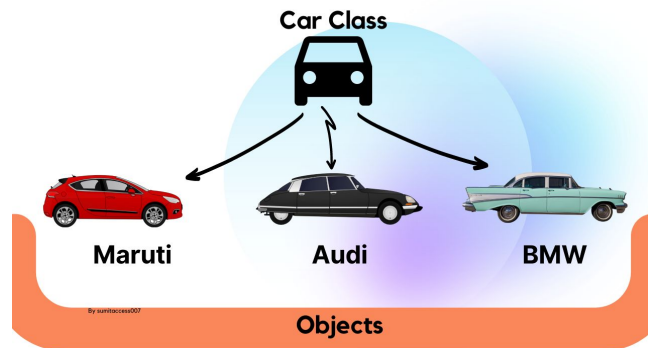
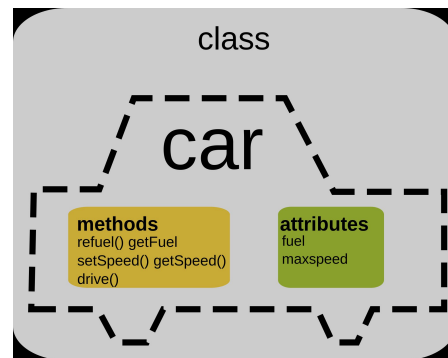
A class :

- It's a blue print .
- It's a design or template.

An Object:

- Its an instance of a class.
- Implementation of a class.

NOTE: Classes are invisible, object are visible



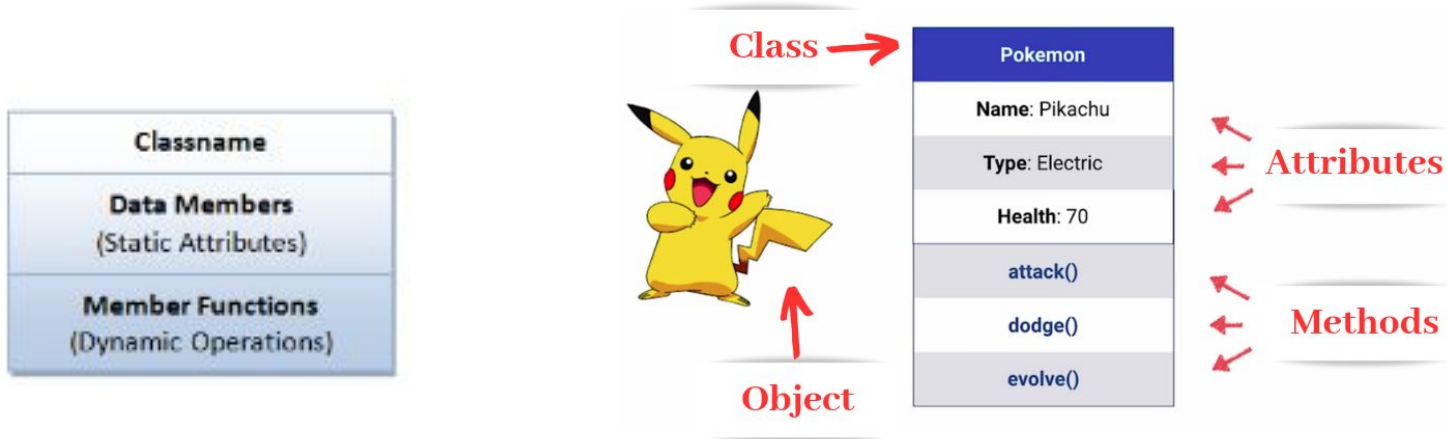
WHAT IS A CLASS?

A class is a 3- compartment box encapsulating data and functions.

Classname (or identifier): identifies the class.

Data Members or Variables: contains the attributes of the class.

Member Functions (or methods, behaviors, operations): contains the dynamic operations of the class.



EXAMPLE OF CLASSES

Classname (Identifier)	Student	Circle
Data Member (Static attributes)	name grade	radius color
Member Functions (Dynamic Operations)	getName() printGrade()	getRadius() getArea()

EXAMPLE OF OBJECTS

Classname	<u>paul:Student</u>	<u>peter:Student</u>
Data Members	name="Paul Lee" grade=3.5	name="Peter Tan" grade=3.9
Member Functions	getName() printGrade()	getName() printGrade()

Two instances of the **Student** class

Each object of a class maintains its own copy of its attributes in memory

CLASS ACTIVITY

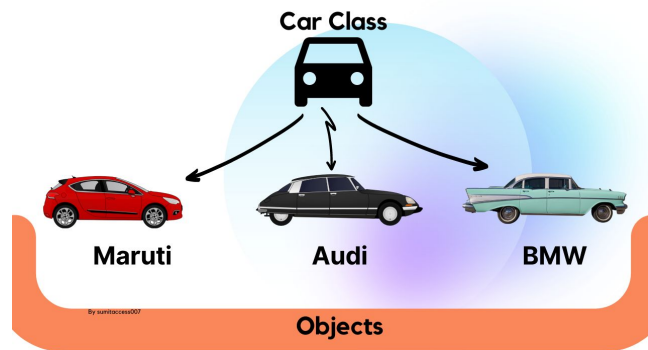
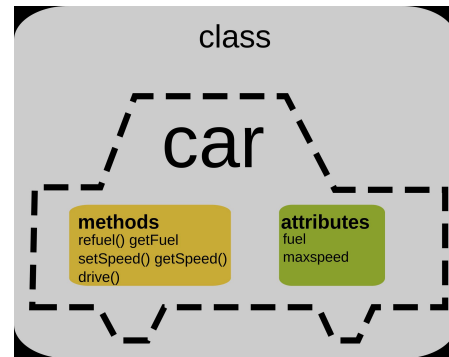
Classes related to point management system.

CLASS

```
class Car  
{
```

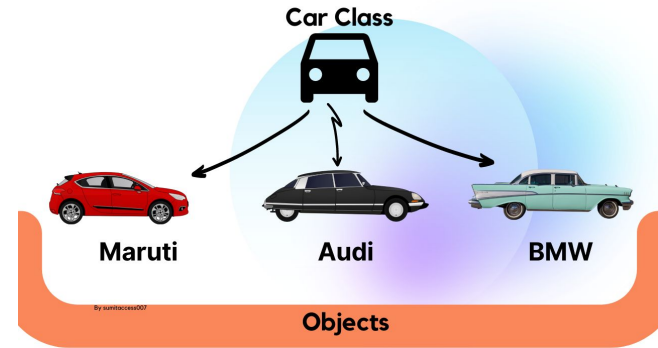
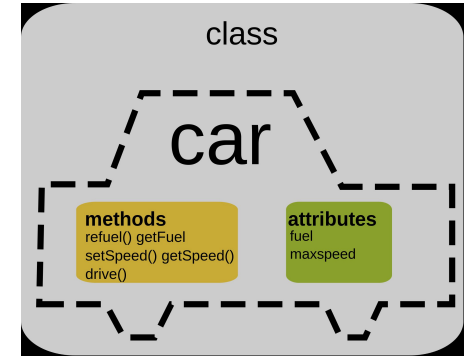
```
void accelerate()  
{ \\ logic for acceleration }
```

```
void brake()  
{ \\ logic for brakes }  
};
```



CLASS

```
class Car
{
    string model;
    int numOfDoors;
    string color;
    void accelerate()
    { \\ logic for acceleration }
    void brake()
    { \\ logic for brakes }
};
```



HOW TO USE THE CAR?

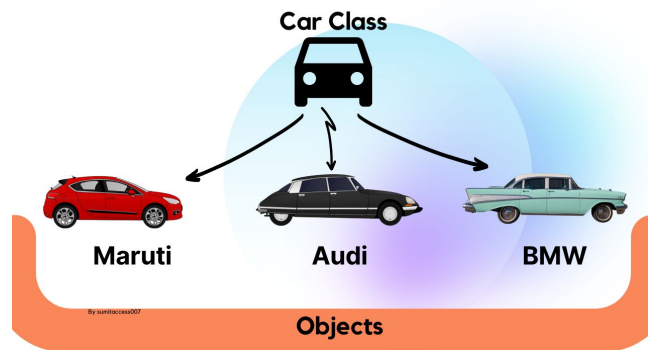
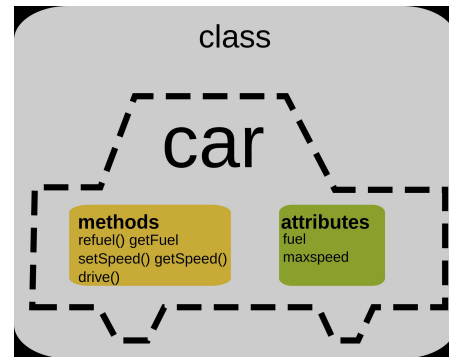
```
int main()
```

```
{
```

```
Car car;
```

```
car.accelerate();
```

```
}
```



CLASS ACTIVITY

Make classes related to environment.

UNIVERSITY

CLASS ACTIVITY

Make classes related to environment.

OFFICE

CLASS ACTIVITY

Make classes related to environment.

HOSPITAL

TYPES OF CLASSES

There are four distinct types of classes which are differentiated based on implementation. They are:

- Stand Alone Classes
- Base Classes
- Derived Classes
- Friend Classes

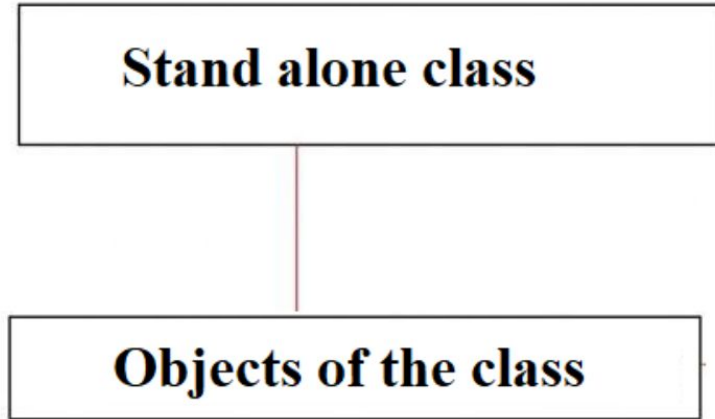
TYPES OF CLASSES

There are six distinct types of classes which are differentiated based on definition and use. They are:

- Stand Alone Classes
- Base Classes
 - Abstract Base Class
 - Concrete Base Class
- Derived Classes
 - Abstract Base Class
 - Concrete Base Class
- Friend Classes

STAND ALONE CLASSES

These classes are neither child classes (i.e they are not derived out of any classes) nor are they parent classes (i.e they do not act as a base class).



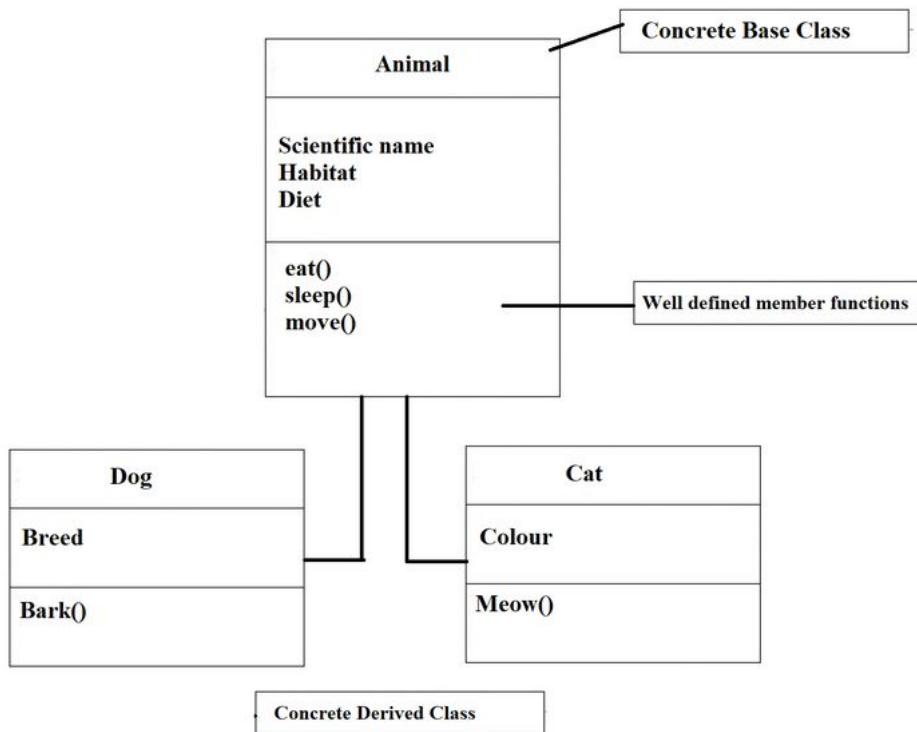
CONCRETE BASE CLASSES AND CONCRETE DERIVED CLASSES

A concrete class has well defined member functions.

Their functions are not **virtual** or **pure virtual**.

A concrete bases class acts as a base for another class to derive from.

A concrete derived class is derived from the existing base class. It inherits the properties of the base class.



CONCRETE BASE CLASSES AND CONCRETE DERIVED CLASSES

CONCRETE BASE CLASS	CONCRETE DERIVED CLASS
Helps to derive or create new classes	Derived out of base class
Also called parent class	Also called child class
Cannot inherit properties and methods of child class	Can inherit properties and methods of parent class
Can be used as stand-alone class	Cannot be used as a stand-alone class

ABSTRACT BASE CLASSES AND ABSTRACT DERIVED CLASSES

Abstract base classes contains pure virtual functions.

It cannot be used to declare the objects of its own.

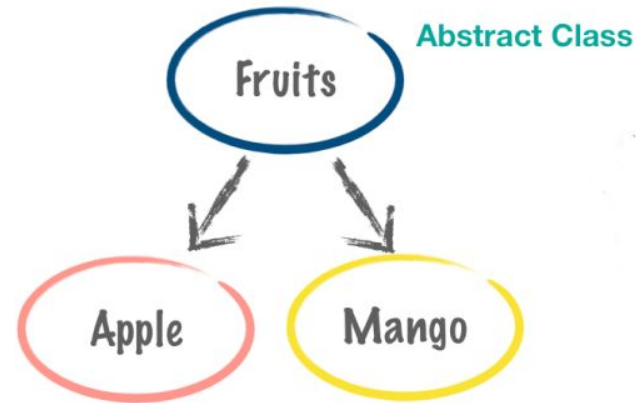
```
Roses are Red  
Violets are Blue  
abstract class Poem  
{  
  //TODO  
}
```



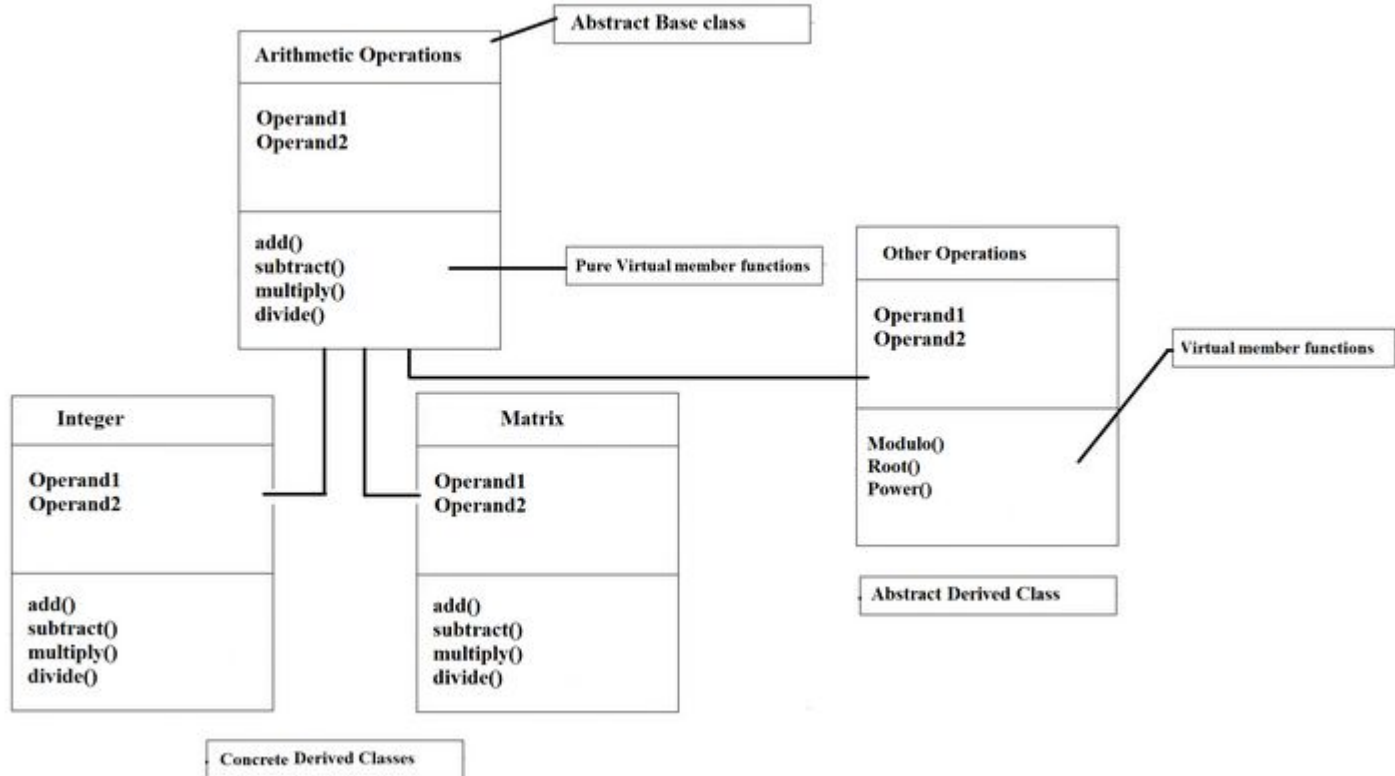
ABSTRACT BASE CLASSES AND ABSTRACT DERIVED CLASSES

Abstract derived class is derived from an abstract base class.

- The remaining functions are all **virtual functions**.
- It is still an abstract class and hence objects of the class cannot be defined.



ABSTRACT BASE CLASSES AND ABSTRACT DERIVED CLASSES



ABSTRACT BASE CLASSES AND ABSTRACT DERIVED CLASSES

ABSTRACT BASE CLASS	ABSTRACT DERIVED CLASS
Helps to derive or create new classes	Derived out of base class
Also called parent class	Also called child class
This class contains pure virtual functions	This class contains virtual functions

FRIEND CLASSES



A friend class can access private and protected members of other classes in which it is declared as a friend.

CLASS ACTIVITY

Name Abstract and Concrete classes name in the following scenario.



0/1- Library



2- Classroom



3- Parents



4- Playground



5- Pen



6- Book



7- Bag



8- Student



9 Teacher

STRUCTURES IN C VS STRUCTURES IN C++

C Structures	C++ Structures
Only data members are allowed, it cannot have member functions.	Can hold both: member functions and data members.
Cannot have static members.	Can have static members.
Cannot have a constructor inside a structure.	Constructor creation is allowed.
Direct Initialization of data members is not possible.	Direct Initialization of data members is possible.
Writing the 'struct' keyword is necessary to declare structure-type variables.	Writing the 'struct' keyword is not necessary to declare structure-type variables.
Do not have access modifiers.	Supports access modifiers .
Only pointers to structs are allowed.	Can have both pointers and references to the struct.
Sizeof operator will generate 0 for an empty structure.	Sizeof operator will generate 1 for an empty structure.
Data Hiding is not possible.	Data Hiding is possible.

CLASSES VS STRUCT

Members of a class are `private` by default and members of struct are `public` by default.

Classes `can be inherited` whereas structures cannot.

Class is `pass-by-reference` and Struct is `pass-by-copy`. Class object is created on the heap memory whereas structure object is created on the stack memory.

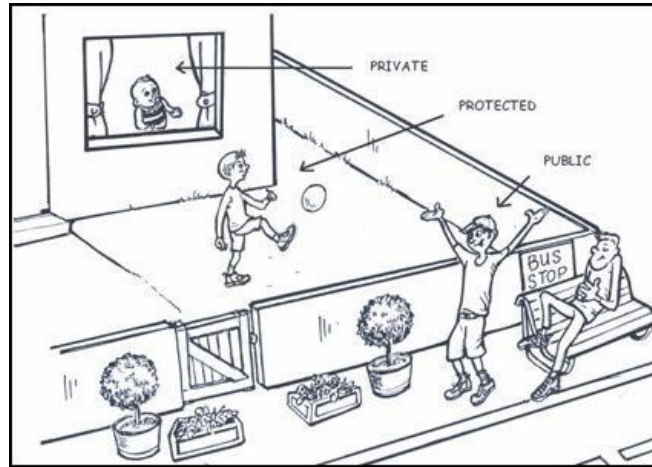
A structure `can't be abstract`, a class can be

CLASSES VS STRUCT

Class	Structure
Members of a class are private by default.	Members of a structure are public by default.
Member classes/structures of a class are private by default.	Member classes/structures of a structure are public by default.
It is declared using the class keyword.	It is declared using the struct keyword.
It is normally used for data abstraction and further inheritance.	It is normally used for the grouping of data

ACCESS MODIFIERS

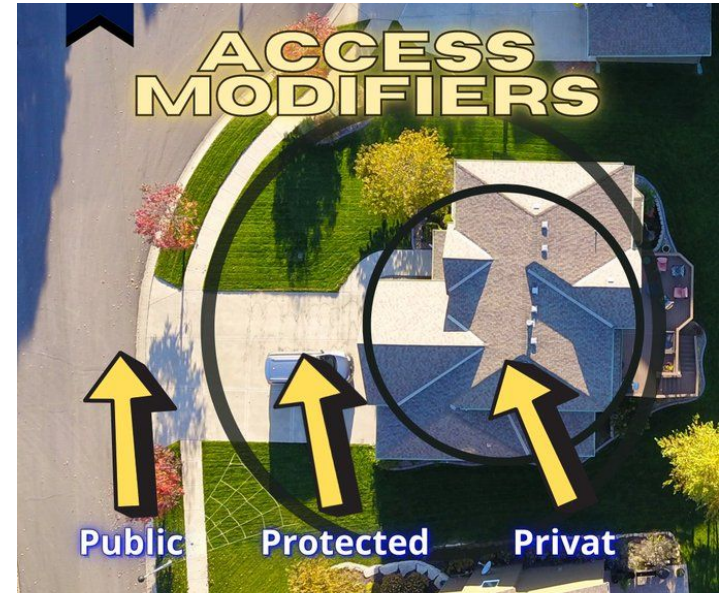
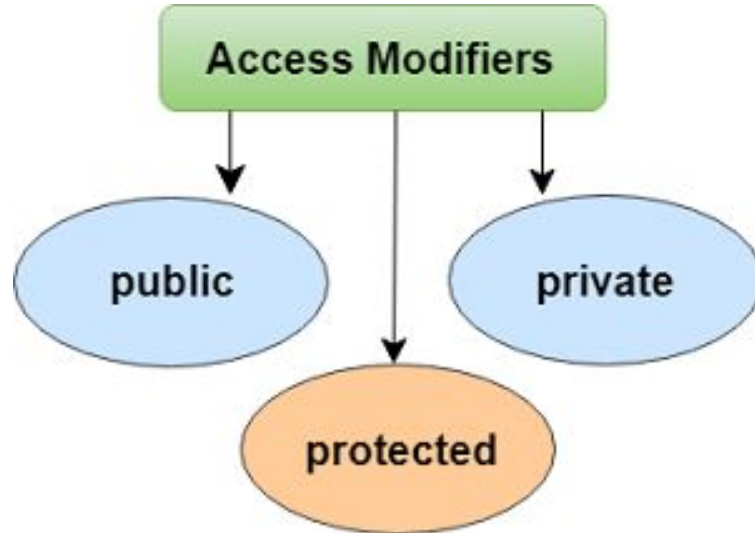
One of the main features of object-oriented programming languages such as C++ is data hiding.












Data hiding refers to restricting access to class members.

ACCESS MODIFIERS

In C++, class members are considered private when no access modifier is used.



ACCESS MODIFIERS VISIBILITY

	Own class	Derived class	Main()
Private			
Protected			
public			

PUBLIC ACCESS MODIFIERS

All the class members (data or functions) declared under the public specifier will be available to everyone.

It can be accessed by any other classes and functions in the program.

We can access the public members of a class directly in the program by using the operator (.) with the name of the object that has been created.

PUBLIC ACCESS MODIFIERS

Output:

```
Radius is: 5.5  
Area is: 94.985
```

```
#include<iostream>  
using namespace std;  
  
// class definition  
class Circle  
{  
    public:  
        double radius;  
  
        double compute_area()  
        {  
            return 3.14*radius*radius;  
        }  
};  
  
// main function  
int main()  
{  
    Circle obj;  
  
    // accessing public data-member outside class  
    obj.radius = 5.5;  
  
    cout << "Radius is: " << obj.radius << "\n";  
    cout << "Area is: " << obj.compute_area();  
    return 0;  
}
```

PRIVATE ACCESS MODIFIERS

The class members declared as private can be accessed only by the public member functions inside the class.

They cannot be accessed by any other object or function outside the class.

Only the member functions also called the friend functions are allowed to access and modify the private data in the class.

PRIVATE ACCESS MODIFIERS

Output:

```
In function 'int main()':
11:16: error: 'double Circle::radius' is private
      double radius;
          ^
31:9: error: within this context
     obj.radius = 1.5;
       ^
```

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

class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        double compute_area()
        {
            // member function can access private
            // data member radius
            return 3.14*radius*radius;
        }
};

// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.radius = 1.5;

    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

PRIVATE ACCESS MODIFIERS

Output:

```
Radius is: 1.5  
Area is: 7.065
```

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

class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        void compute_area(double r)
        {
            // member function can access private
            // data member radius
            radius = r;

            double area = 3.14*radius*radius;

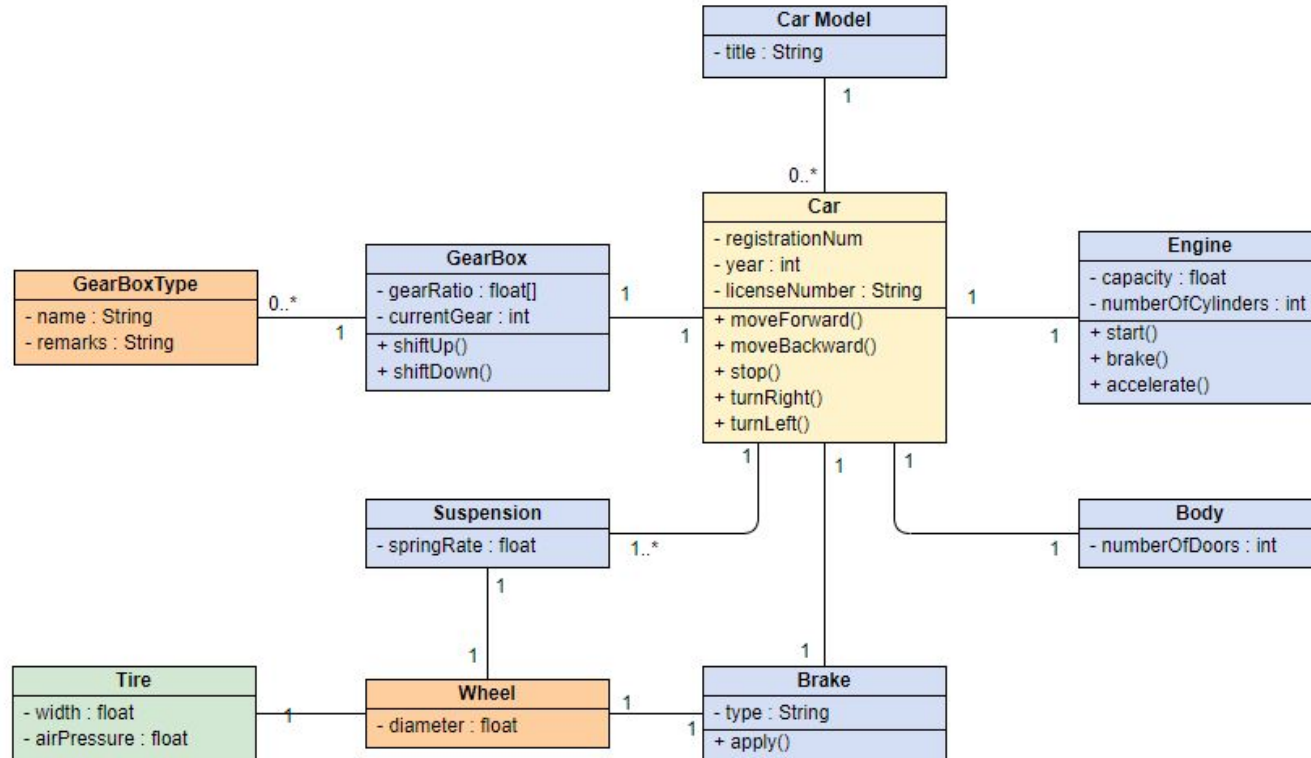
            cout << "Radius is: " << radius << endl;
            cout << "Area is: " << area;
        }
};

// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.compute_area(1.5);

    return 0;
}
```


ACCESS MODIFIERS IN CLASS DIAGRAM



REFERENCES

<https://javascript.plainenglish.io/what-are-javascript-programming-paradigms-3ef0f576dfdb>

<https://www.javatpoint.com/java-oops-concepts>

<https://iq.opengenus.org/types-of-classes-in-cpp/>

<https://www.geeksforgeeks.org/difference-c-structures-c-structures/>

<https://inprogrammer.com/access-modifiers-in-cpp/>

<https://www.youtube.com/watch?app=desktop&v=IPGbZJ8i7ss>