Lecture 18

M M Imran

Object Oriented Programming

- OOPs, or Object-oriented programming is an approach or a programming pattern where the programs are structured around objects rather than functions and logic.

- The core of the pure object-oriented programming is to create an object, in code, that has certain properties and methods.

- While designing C++ modules, we try to see whole world in the form of objects.

Object Oriented Programming

- For example a car is an object which has certain properties such as color, number of doors, and the like. It also has certain methods such as accelerate, brake, and so on.

- The prime purpose of C++ programming was to add object orientation to the C programming language.

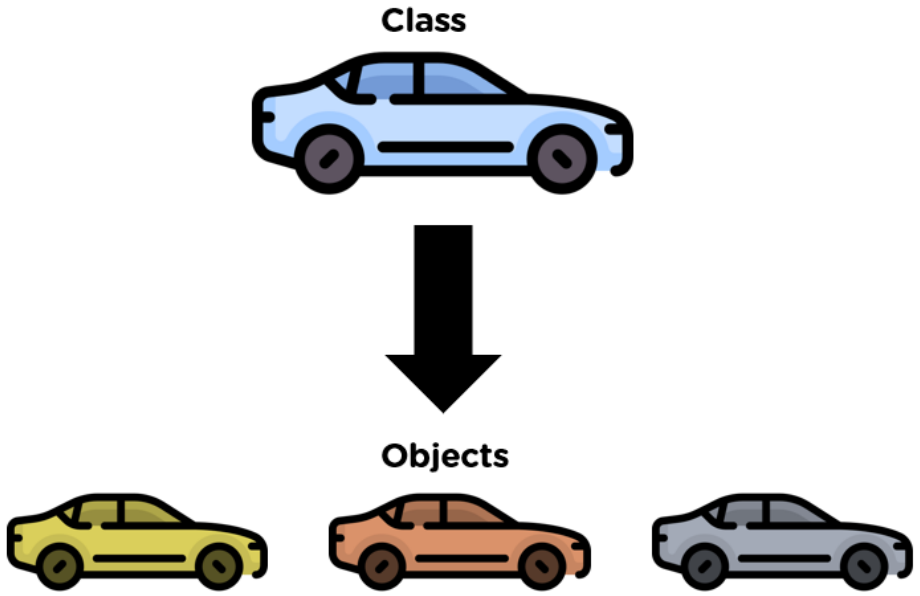
- OOP language allows to break the program into the bit-sized problems that can be solved easily

Basic Object-Oriented Programming Concepts

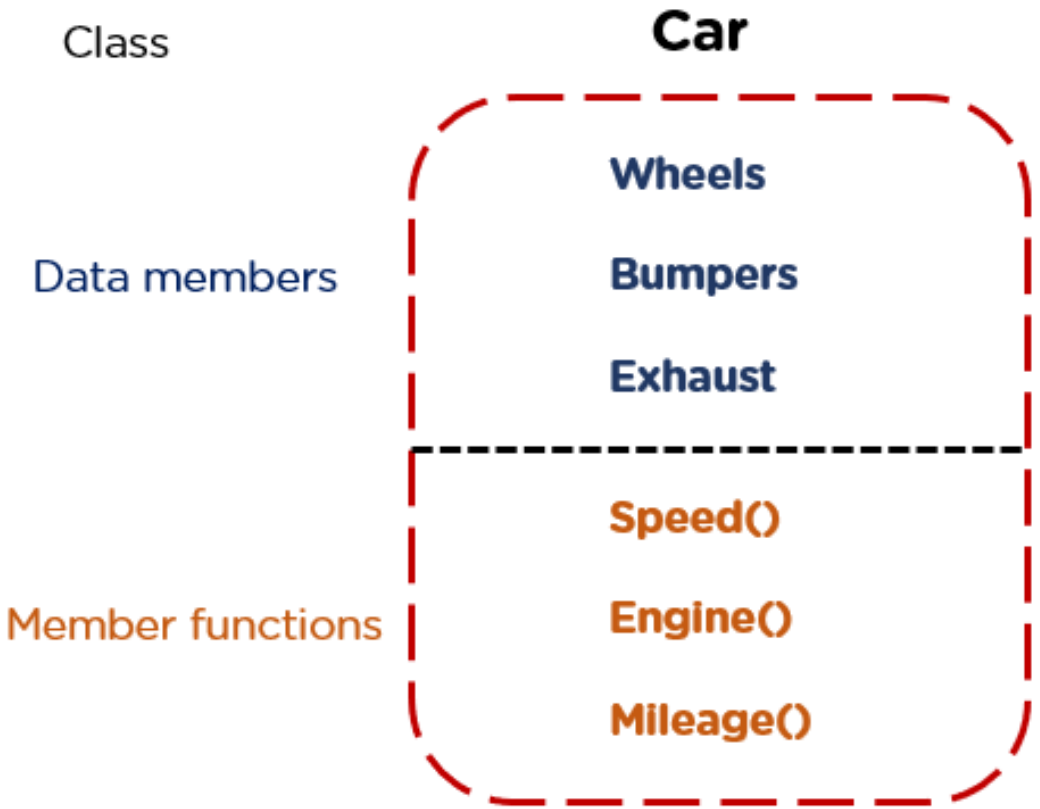
- Classes & Objects
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Object: An Object can be defined as an entity that has a state and behavior, or in other words, anything that exists physically in the world is called an object. It can represent a dog, a person, a table, etc.

Class: Class can be defined as a blueprint of the object. It is basically a collection of objects which act as building blocks.



- A class contains data members (variables) and member functions. These member functions are used to manipulate the data members inside the class.



Abstraction

- Abstraction helps in the data hiding process. It helps in displaying the essential features without showing the details or the functionality to the user.

- It avoids unnecessary information or irrelevant details and shows only that specific part which the user wants to see.

Encapsulation

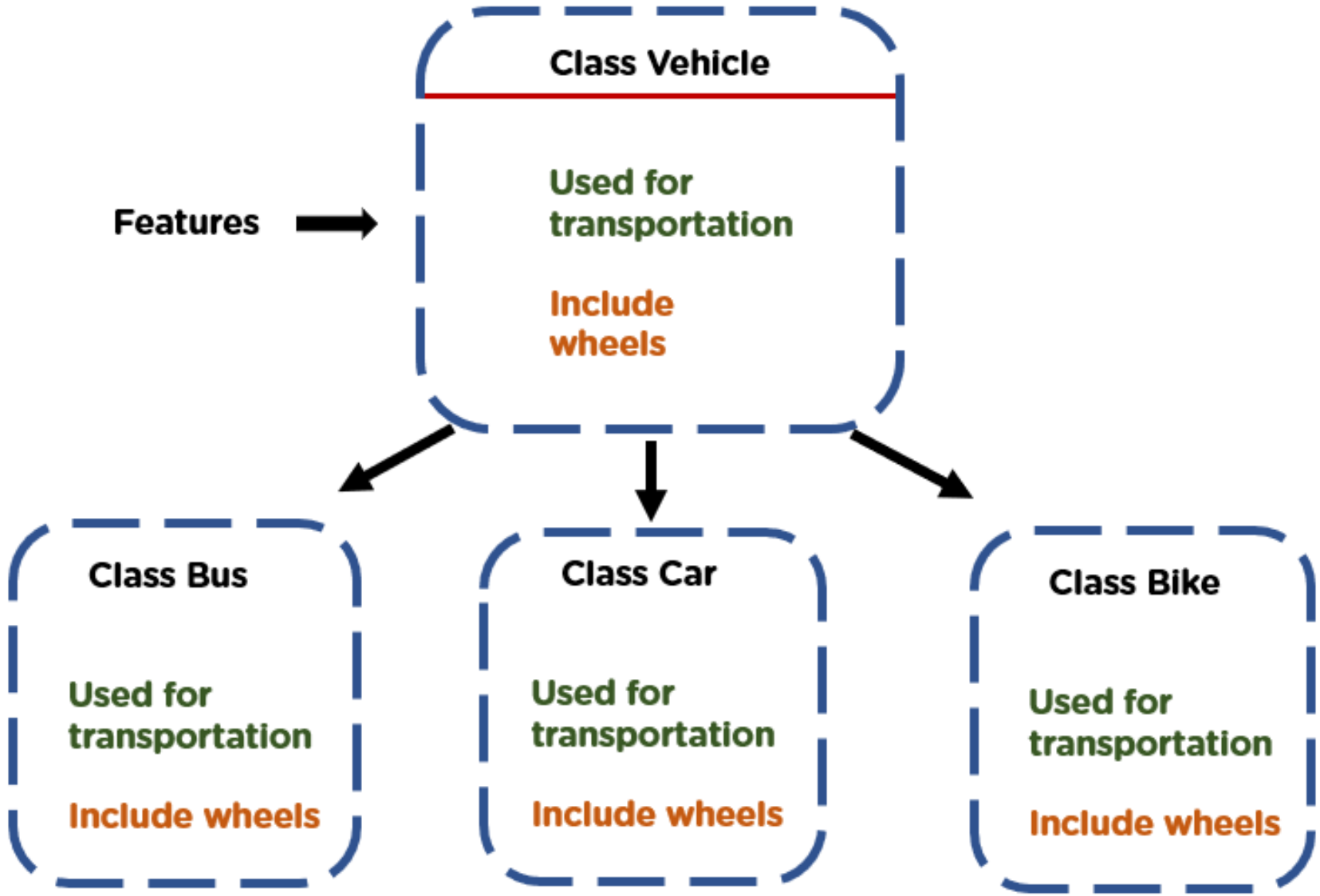
- The wrapping up of data and functions together in a single unit is known as encapsulation.
- It can be achieved by making the data members' scope private and the member function’s scope public to access these data members.
- Encapsulation makes the data non-accessible to the outside world.

Inheritance

- Inheritance is the process in which two classes have an is-a relationship among each other and objects of one class acquire properties and features of the other class.

- The class which inherits the features is known as the child class, and the class whose features it inherited is called the parent class.

- For example, Class Vehicle is the parent class, and Class Bus, Car, and Bike are child classes.


Features
Class Vehicle
Used for transportation
Include wheels
Class Bus
Used for transportation
Include wheels
Class Car
Used for transportation
Include wheels
Class Bike
Used for transportation
Include wheels

```cpp
#include <iostream>

using namespace std;

class Parent
{
    public:
    string name1="Harley";
};

class Child: public Parent
{
    public:
    string name2="Davidson";
};

int main()
{
    Child ch;
    cout << ch.name1 + " " + ch.name2;

    return 0;
}
```

```
Harley Davidson
```

Here are two classes named parent and child. In line 12, the child class inherits the parent class.

Polymorphism

- Polymorphism means many forms. It is the ability to take more than one form.

- It is a feature that provides a function or an operator with more than one definition.

- It can be implemented using function overloading, operator overload, function overriding, virtual function.

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

class Animal {
  public:
    void speed() {
      cout << "Who is more faster\n" ;
    }
};

class Cheetah : public Animal {
  public:
    void speed() {
      cout << "cheetah says im faster \n" ;
    }
};

class Dolphin : public Animal {
  public:
    void speed() {
      cout << "Dolphin says im faster  \n" ;
    }
};
```

```cpp
int main() {
  Animal a;
  Cheetah c;
  Dolphin d;

  a.speed();
  c.speed();
  d.speed();
  return 0;
}
```

```
Who is more faster
cheetah says im faster
Dolphin says im faster
```

Advantages of OOPs

- OOPs provide reusability to the code and extend the use of existing classes.
- In OOPs, it is easy to maintain code as there are classes and objects, which helps in making it easy to maintain rather than restructuring.
- It also helps in data hiding, keeping the data and information safe from leaking or getting exposed.
- Object-oriented programming is easy to implement.

More Examples

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

// base class
class Animal {

   public:
    void eat() {
        cout << "I can eat!" << endl;
    }

    void sleep() {
        cout << "I can sleep!" << endl;
    }
};

// derived class
class Dog : public Animal {

   public:
    void bark() {
        cout << "I can bark! Woof woof!!" << endl;
    }
};
```

```cpp
int main() {
    // Create object of the Dog class
    Dog dog1;

    // Calling members of the base class
    dog1.eat();
    dog1.sleep();

    // Calling member of the derived class
    dog1.bark();

    return 0;
}
```

Output

```
I can eat!
I can sleep!
I can bark! Woof woof!!
```

Protected Members

- The access modifier protected is especially relevant when it comes to C++ inheritance.

- Like private members, protected members are inaccessible outside of the class. However, they can be accessed by derived classes and friend classes/functions.

- We need protected members if we want to hide the data of a class, but still want that data to be inherited by its derived classes.

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

// declare parent class
class Sample {
    // protected elements
   protected:
    int age;
};

// declare child class
class SampleChild : public Sample {

   public:
    void displayAge(int a) {
        age = a;
        cout << "Age = " << age << endl;
    }

};
```

```cpp
int main() {
    int ageInput;

    // declare object of child class
    SampleChild child;

    cout << "Enter your age: ";
    cin >> ageInput;

    // call child class function
    // pass ageInput as argument
    child.displayAge(ageInput);

    return 0;
}
```

Output

```
Enter your age: 20
Age = 20
```

Access Modifiers Summary

- **public** elements can be accessed by all other classes and functions.
- **private** elements cannot be accessed outside the class in which they are declared, except by friend classes and functions.
- **protected** elements are just like the private, except they can be accessed by derived classes.

Specifiers	Same Class	Derived Class	Outside Class
`public`	Yes	Yes	Yes
`private`	Yes	No	No
`protected`	Yes	Yes	No

```cpp
// base class
class Animal {

   private:
    string color;

   protected:
    string type;

   public:
    void eat() {
        cout << "I can eat!" << endl;
    }

    void sleep() {
        cout << "I can sleep!" << endl;
    }

    void setColor(string clr) {
        color = clr;
    }

    string getColor() {
        return color;
    }
};
```

```cpp
// derived class
class Dog : public Animal {

   public:
    void setType(string tp) {
        type = tp;
    }

    void displayInfo(string c) {
        cout << "I am a " << type << endl;
        cout << "My color is " << c << endl;
    }

    void bark() {
        cout << "I can bark! Woof woof!!" << endl;
    }
};
```

```cpp
int main() {
    // Create object of the Dog class
    Dog dog1;

    // Calling members of the base class
    dog1.eat();
    dog1.sleep();
    dog1.setColor("black");

    // Calling member of the derived class
    dog1.bark();
    dog1.setType("mammal");

    // Using getColor() of dog1 as argument
    // getColor() returns string data
    dog1.displayInfo(dog1.getColor());

    return 0;
}
```

```
I can eat!
I can sleep!
I can bark! Woof woof!!
I am a mammal
My color is black
```

Considerations

- Here, the variable type is protected and is thus accessible from the derived class Dog. We can see this as we have initialized type in the Dog class using the function setType().
- On the other hand, the private variable color cannot be initialized in Dog.

```
class Dog : public Animal {

    public:
      void setColor(string clr) {
          // Error: member "Animal::color" is inaccessible
          color = clr;
      }
};
```

Considerations

- Also, since the protected keyword hides data, we cannot access type directly from an object of Dog or Animal class.

```
// Error: member "Animal::type" is inaccessible
dog1.type = "mammal";
```