Object Oriented Programming

Tushar B. Kute,
http://tusharkute.com

Object Oriented Programming

- Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.
- It simplifies the software development and maintenance by providing some concepts:
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation

What is Object Oriented Programming?

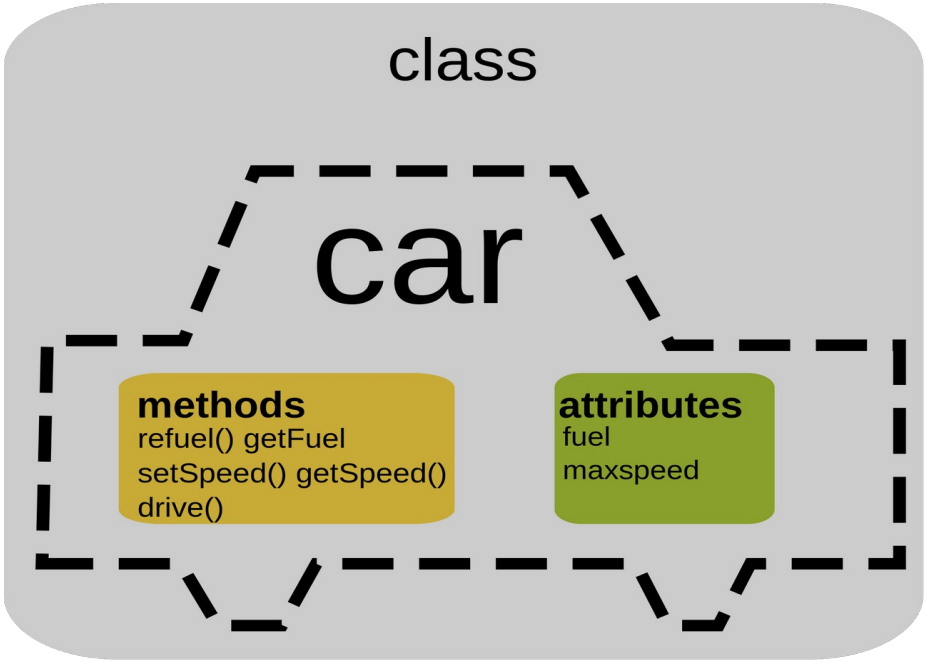
- Object-oriented Programming, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.

What is Object Oriented Programming?

- Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc.
- OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.

Class

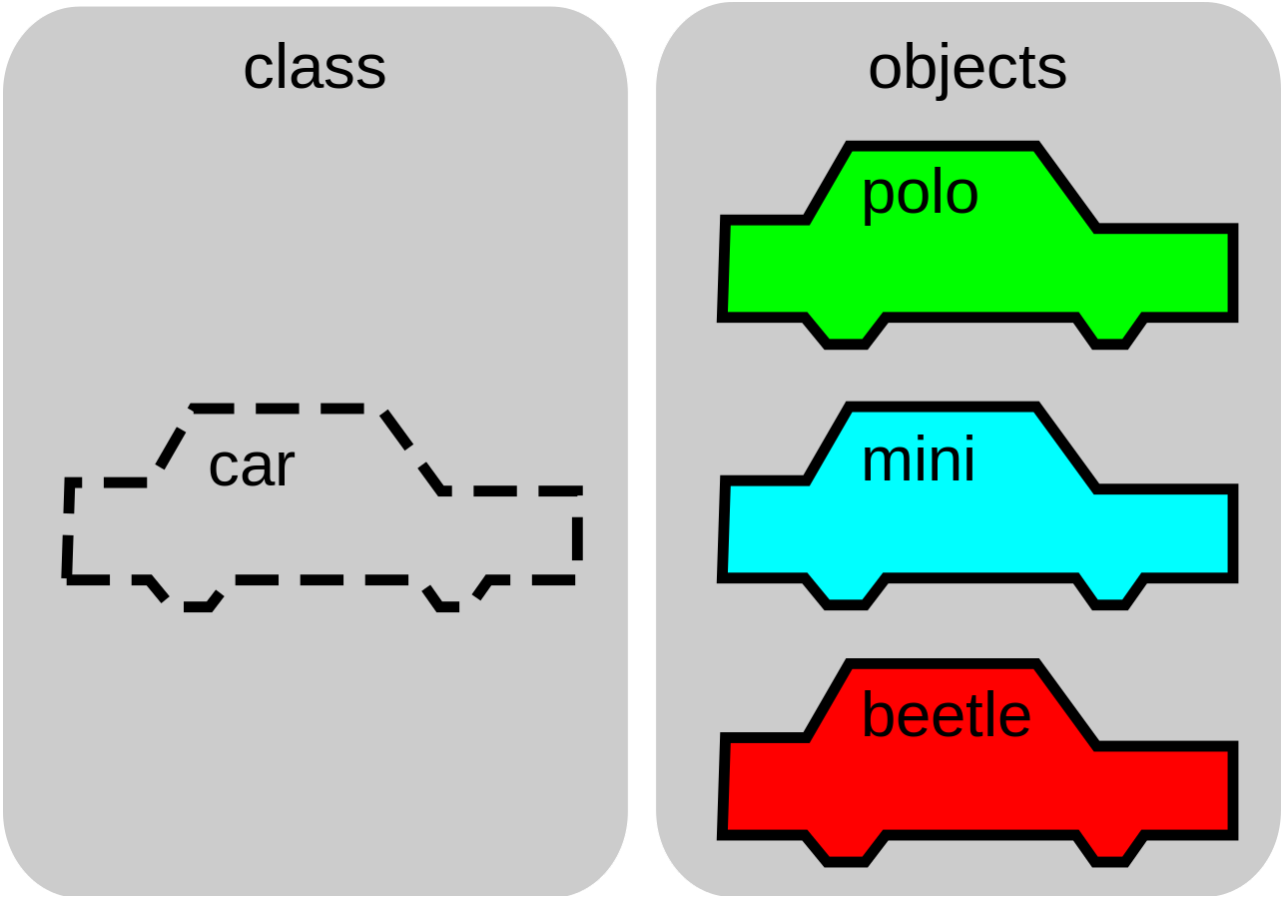
Classes are used to create new user-defined data structures that contain arbitrary information about something. In the case of an car, we could create an Car() class to track properties about the Car like the fuel and maxspeed.



Objects

- While the class is the blueprint, an instance is a copy of the class with actual values, literally an object belonging to a specific class.
- Put another way, a class is like a form or questionnaire.
- It defines the needed information. After you fill out the form, your specific copy is an instance of the class; it contains actual information relevant to you.

Objects



OOP Features

- Reduction in the complexity
- Importance of data
- Creation of new data structure
- Data Hiding
- Characterization of the objects
- Communication among objects
- Extensibility
- Bottom-up programming approach

OOP Concepts

- Class
- Object
- Data Hiding / abstraction / encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

Inheritance

- When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
  - Sub class - Subclass or Derived Class refers to a class that receives properties from another class.
  - Super class - The term "Base Class" or "Super Class" refers to the class from which a subclass inherits its properties.
  - Reusability - As a result, when we wish to create a new class, but an existing class already contains some of the code we need, we can generate our new class from the old class thanks to inheritance. This allows us to utilize the fields and methods of the pre-existing class.

Polymorphism

- When one task is performed by different ways i.e. known as polymorphism.
- For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.
- Different situations may cause an operation to behave differently.
- The type of data utilized in the operation determines the behavior.

Abstraction

- Hiding internal details and showing functionality is known as abstraction.
- Data abstraction is the process of exposing to the outside world only the information that is absolutely necessary while concealing implementation or background information.
- For example: phone call, we don't know the internal processing.
- In C++, we use abstract class and interface to achieve abstraction.

Encapsulation

- Binding (or wrapping) code and data together into a single unit is known as encapsulation.
- For example: capsule, it is wrapped with different medicines.
- Encapsulation is typically understood as the grouping of related pieces of information and data into a single entity.
- Encapsulation is the process of tying together data and the functions that work with it in object-oriented programming.

Creating class and object

- In C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
- In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.
- Object is a runtime entity, it is created at runtime.

Creating class and object

- In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.
- Let's see an example of C++ class that has three fields only.

```
class Student
{
    public:
    int id;  //field or data member
    float salary; //field or data member
    String name;//field or data member
}
```

Creating class and object

- C++ Class Example: Initialize and Display data through method

Constructor

- In C++, constructor is a special method which is invoked automatically at the time of object creation.
- It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.
- In brief, A particular procedure called a constructor is called automatically when an object is created in C++.
- In general, it is employed to create the data members of new things.
- In C++, the class or structure name also serves as the constructor name. When an object is completed, the constructor is called. Because it creates the values or gives data for the thing, it is known as a constructor.

Constructor

- Inside the class Syntax:
  - <class-name> (list-of-parameters) { // constructor definition }
- The following syntax is used to define a constructor outside of a class:
  - <class-name>: :<class-name> (list-of-parameters) { // constructor definition}

Constructor

- Constructors lack a return type since they don't have a return value.
- There can be two types of constructors in C++.

  Default constructor

  Parameterized constructor
- C++ Default Constructor
  - A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Constructor

- The constructor has the same name as the class it belongs to.
- Although it is possible, constructors are typically declared in the class's public section. However, this is not a must.
- Because constructors don't return values, they lack a return type.
- When we create a class object, the constructor is immediately invoked.
- Overloaded constructors are possible.
- Declaring a constructor virtual is not permitted.
- One cannot inherit a constructor.
- Constructor addresses cannot be referenced to.
- When allocating memory, the constructor makes implicit calls to the new and delete operators.

Destructor

- An equivalent special member function to a constructor is a destructor.
- The constructor creates class objects, which are destroyed by the destructor.
- The word "destructor," followed by the tilde () symbol, is the same as the class name. You can only define one destructor at a time.

Destructor

- One method of destroying an object made by a constructor is to use a destructor. Destructors cannot be overloaded as a result.
- Destructors don't take any arguments and don't give anything back. As soon as the item leaves the scope, it is immediately called.
- Destructors free up the memory used by the objects the constructor generated. Destructor reverses the process of creating things by destroying them.

Destructor

- The language used to define the class's destructor

  ~ <class-name>()

  {

  }

- The language used to define the class's destructor outside of it

  <class-name>: : ~ <class-name>(){}

Encapsulation

- All C++ programs are composed of the following two fundamental elements –
  - Program statements (code) – This is the part of a program that performs actions and they are called functions.
  - Program data – The data is the information of the program which gets affected by the program functions.

Encapsulation

- Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.
- Data encapsulation led to the important OOP concept of data hiding.
- Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

Encapsulation

- C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called classes.
- We already have studied that a class can contain private, protected and public members. By default, all items defined in a class are private.

Encapsulation

- For example –

```
class Box {
  public:
    double getVolume(void) {
      return length * breadth * height;
    }

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

Inheritance

- One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- This also provides an opportunity to reuse the code functionality and fast implementation time.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.
- This existing class is called the base class, and the new class is referred to as the derived class.

Inheritance

- A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes.
- To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form –

  class derived-class: access-specifier base-class
- Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Inheritance

- A derived class can access all the non-private members of its base class.
- Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.
- We can summarize the different access types according to - who can access them in the following way –

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Inheritance

- A derived class inherits all base class methods with the following exceptions –
  - Constructors, destructors and copy constructors of the base class.
  - Overloaded operators of the base class.
  - The friend functions of the base class.

Multiple Inheritance

- A C++ class can inherit members from more than one class and here is the extended syntax –

  class derived-class: access baseA, access baseB....

- Where access is one of public, protected, or private and would be given for every base class and they will be separated by comma as shown above.

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com