OOPs Principles

# Abstraction

* Know how to use but don’t know internal mechanism (you know name and application of function but don’t know how it works)

# Encapsulation – Data Hiding

# Inheritance

* One design can be followed or inherited by another.
* Ex. You design one car and you will use features of this car to make its succeeding model.

**Notes:**

**An important thing is to be noted is that protected members can’t be accessed by any object directly.**

**Private members cannot be accessed directly by any object (except its member functions).**

**However a protected members are inherited to derived class and also can be accessed by derived class member functions;**

# Polymorphism

* Function overloading
* Function overriding

Note:

You can access **data only by functions**.

Functions should be public and data should be private.

It will avoid mishandling of data.

* Classification by classes.
* Object is an instance of class.
* Ex. Toyota is object of cars.
* Cars class is a design or blueprint of Toyota.
* Class contains data and functions (also called behavior or methods).
* Class can be used to create user defined data type.
* Everything inside class is private by default.

# Pointer to an object:

* We can access class functions using arrow operator (->) by pointer.

# Data Hiding:

* You know how to drive car but don’t know internal mechanism.
* Make data members private and functions public (Good practice).
* Accessor: function which is reading value.
* Mutator: functions which write to variable.

# Constructors:

* Data members will contain garbage value if not initialized.
* It will be created and executed at the time when object is declared.
* Compiler will create constructor (Default constructor) if there isn’t any user defined constructor.
* Types: Default, Non-parameterized, parameterized and copy constructor.
* Deep copy constructor: Be careful with it when it comes to dynamic memory allocation.

# Types of functions in a class:

* Non parameterized constructor
* Parameterized constructor
* Copy constructor
* Accessor: to read value of variable.
* Mutator: setData (), setLength ().
* Facilitators: will perform any calculation like area, perimeter.
* Inspector function: To check a possibility.
* Destructor : ~class\_name.
* Note: Declare only prototypes of functions inside the class.

# Scope resolution operator (::):

* return\_type class\_name::function\_name(){}

# Inline function:

* No separate block will be created for inline function in memory area.
* Functions defined inside class are inline by default.
* Functions defined outside class are non-inline by default.
* Write inline (key word) before the name of function to make it inline.

# Structure vs Class:

|  |  |
| --- | --- |
| **structure** | **class** |
| By default public | private |
| Functions cannot be defined inside structure | Functions can be defined inside the class |

Polymorphism:

Same thing but performs various tasks.

# Operator overloading

* Friend function: doesn’t belong to class but

# Function overriding:

* Same functions but different definitions in same scope or class.

# Function hiding:

* Same functions but different definitions in different classes or scopes.
* If function is called by an object of child class then first priority will be given to the function declared in child class, if function is not present in child class then parent’s class function is called.
* This is called function hiding.

# Virtual function:

* Member function of base class redefined in child class (using virtual keyword).
* Used in late binding.
* Base class pointer contains address of the derived class object but executes definition of base class function.
* ‘Virtual’ keyword helps to determine which function to be invoked.
* Virtual function must be defined in the base class, even though it is not used.
* Virtual destructor is allowed but virtual constructor is not allowed.
* **Pure virtual function**
* A virtual function is not used for performing any task. It only serves as a placeholder.
* When the function has no definition, such function is known as "**do-nothing**" function.
* The "**do-nothing**" function is known as a **pure virtual function**. A pure virtual function is a function declared in the base class that has no definition relative to the base class.
* A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.
* The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.
* **Definition of pure virtual function : virtual** **void** display() = 0;

Abstract classes

Abstract classes are the way to achieve abstraction in C++. Abstraction in C++ is the **process to hide the internal details and showing functionality only**. Abstraction can be achieved by two ways:

1. **Abstract class**
2. **Interface**