**TypeScript OOP**

**OOPS: - Object Oriented Programming Structure**

**Constructor**: - it is a special method of the class which meant for specifying action for the object. i.e. initializing properties of the class at the time of object creation.

Imp. Points: -

1. It will be created with “constructor” keyword as follows

constructor([parameters])

{

Body of the constructor

}

1. It will be executed whenever object is created for that class.
2. It can take parameters but it won’t return any value.
3. Whenever constructor is taking parameters, for that parameters values has to be passed at the time of object creation.

**Inheritance: -** it is the process of getting the features from parent item and applying into child item.

Purpose of the inheritance is, reusability of the classes. i.e. using one class data into another specific class

**Imp. Points: -**

1. Here parent item is called as Base Class and child item is called as Derived Class.
2. Inheritance will be done from base class to derived class only.
3. To inherit a base class into derived class “extends” keyword is used.
   1. **class <derived class> extends <base class>**
4. To active base class properties into derived class, base class properties should **protected** access modifier.
5. No need to create any object for base class, just create object for derived classes then call the methods from both base and derived classes.

**Types of inheritances: -**

Simple Inheritance/Single Inheritance

Multiple Inheritance

Multi-Level Inheritance

Hierarchical inheritance

Hybrid inheritance

Single Inheritance :- There is Single Base Class and Single Derived Class

Object

Derived Class

Base Class

Multi-Level Inheritance: - multiple base classes and derived classes are arranged in a chain linking process.

Derived -2

Derived -1

Base Class

Object

class base

class derived1 extends base

class derived2 extends derived1

Multiple Inheritance :- No of Base classes are inheriting one derived class.

Note:- it cannot be implemented directly using classes, because only one parent class will be inherited to the child class.

To Achieve this, **interface** concept is used.

Base Class - 2

Base Class - 1

Derived

Object

Hierarchical inheritance: - One base class is inheriting multiple derived classes.

Base Class

Object

Object

Derived - 2

Derived - 1

Hybrid Inheritance: - the combination of any two types of inheritances framing as one inheritance

Base Class - 1

Base Class - 2

Derived - 1

Derived - 2

Object

**Interface: -**

It is also one type of class which consist only declarations of the methods and properties.

The purpose of the interface is to implement multiple inheritance. So that every interface works like a parent / base class, every method declared inside the interface should have definition in it’s child/derived class.

Interface <iface name>

{

Property declaration;

Methodname():returntype;

}

Once interface is created, to inherit that interface to the class, “implements” keyword is used.

Class <classname> implements <interface name>

{

}

**Enumerations: -**

it is a collection of constants grouping under one unit.

It acts as one of the user defined data types.

Once enumeration is created, it can be assigned to a variable or a property.

Propertyname:enumerationname;

The constants defined within the enumeration will have an index value starts from 0, …..

To create an enumeration, enum keyword is used.

enum <enumeration\_name>

{

Const-1, Const-2, ……

}

enum courses

{

Java, dotnet, python, angular, react js

}

Ex: -

propertyname : enumeration type

for assigning value :-

this.propname = enumerationame.value;

to print value :-

enumerationname[this.prop\_name]

**Modules: -**

A module is an typescript class file(s) where it allows to export from one program to another program.

Purpose of this concept is, export one class into another class within the application.

While creating a class, we have to use “export” keyword for class creation.

export class <classnme>

{

Body of the class

}

Where ever using the class, there we have to import the class as follows

import { classname, ..} from “./filename”;

class <classname>

{

Body of the class

}

To represent :-

“./” will represent currently working folder

“./subfoldername/filename” will represent sub folder of the currently work folder.

“../foldername” will represent parent folder.