**Type Script Class**

working with class

classes are the fundamental entities which are used to create reusable components.

A class definition can include the following −

**Fields −** A field is any variable declared in a class. Fields represent data pertaining to objects

**Constructors −** Responsible for allocating memory for the objects of the class

**Functions** − Functions represent actions an object can take. They are also at times referred to as methods

**Syntax to declare a class**

A class keyword is used to declare a class in TypeScript.

We can create a class with the following syntax:

class <class\_name>{

field;

method;

}

**Creating an object of class**

A class creates an object by using the new keyword followed by the class Name

The new keyword allocates memory for object creation at runtime.

All objects get memory in heap memory area.

let object\_name = new class\_name(parameter)

class Employee {

empCode: number;

empName: string;

constructor(code: number, name: string) {

this.empName = name;

this.empCode = code;

}

getSalary() : number {

return 10000;

}

}

class Car {

//field

engine:string;

//constructor

constructor(engine:string) {

this.engine = engine

} //

function

disp():void {

console.log("Function displays Engine is : "+this.engine)

}

} //

create an object

var obj = new Car("abctest")

//access the field

console.log("Reading attribute value Engine as : "+obj.engine)

//access the function

obj.disp()

**Class Inheritance**

Inheritance is the ability of a program to create new classes from an existing class.

The class that is extended to create newer classes is called the parent class/super class. The newly created classes are called the child/sub classes.

A class inherits from another class using the ‘extends’ keyword. Child classes inherit all properties and methods except private members and constructors from the parent class.

class child\_class\_name extends parent\_class\_name

class Shape {

Area:number

constructor(a:number) {

this.Area = a

}

}

class Circle extends Shape {

disp():void {

console.log("Area of the circle: "+this.Area)

}

}

var obj = new Circle(223);

obj.disp()

**Inheritance can be classified as −**

**Single** − Every class can at the most extend from one parent class

**Multi-level** − The following example shows how multi-level inheritance works.

**Multiple** − A class can inherit from multiple classes.

TypeScript doesn’t support multiple inheritance.

Multiple Inheritance

Hierarchical Inheritance

Hybrid Inheritance

class Car {

Color:string

constructor(color:string) {

this.Color = color

}

}

class Audi extends Car {

Price: number

constructor(color: string, price: number) {

super(color);

this.Price = price;

}

display():void {

console.log("Color of Audi car: " + this.Color);

console.log("Price of Audi car: " + this.Price);

}

}

let obj = new Audi(" Black", 8500000 );

obj.display();

**typecast**

which allow you to convert a variable from one type to another type.

Type castings allow you to convert a variable from one type to another.

let str:any="234";

let s = <string>str;

let sq = str as number;

console.log(s);

console.log(sq);

**TypeScript Type Assertion**

is a mechanism which tells the compiler about the type of the variable.

Type assertion works like typecasting,

but it does not perform type checking or restructuring of data

The typecasting comes with runtime support,

whereas type assertion has no impact on runtime.

type assertions are purely a compile-time construct

TypeScript provides two ways to do Type Assertion. They are

Using Angular Bracket <>

Using as keyword

let ecode:any=344;

let empcode=<number>ecode;

let emmpc = ecode as string;

console.log(typeof(empcode));

**Type Assertion with Object**

Similarly, we might have a situation where we have an object that has been

declared without any properties yet

let employee = { };

employee.name = "John";

employee.code = 123;

interface Employee {

name: string;

code: number;

}

let employee = <Employee> { };

employee.name = "John";

employee.code = 123;

**static properties**

The static members can be defined by using the keyword static.

Can be accessed without creating instance of a class(object)

Called using the class name

To create helper or utility methods(they do not depend the data,they depend only class)

The static members of a class are accessed using the class name and dot notation,without creating an object

class Circle {

static pi: number = 3.14;

let pi1:number=90;

static calculateArea(radius:number) {

return this.pi \* radius \* radius;

}

}

console.log(Circle.pi)

console.log(Circle.calculateArea(5))

let ss= new Circle()

console.log(ss)

**abstract class**

Abstract class is a class which may have some unimplemented methods.

These methods are called abstract methods.

abstract class in Typescript using the abstract keyword.

Abstract classes are mainly for inheritance

We cannot create an instance of an abstract class.

An abstract class typically includes one or more abstract methods or property

declarations.

abstract class Person {

abstract name: string;

display(): void{

console.log(this.name);

}

}

class Employee extends Person {

name: string;

empCode: number;

constructor(name: string, code: number) {

super();

this.empCode = code;

this.name = name;

}

}

let emp: Person = new Employee("James", 100);

emp.display();