St. Francis Institute of Technology Borivli (West), Mumbai-400103
(Autonomous Institute)
Department of Information Technology

Academic Year: 2024-25
Class: TE-ITA/B **Semester:** VI

Subject: Web Lab

Experiment – 3: Inheritance and Access modifiers in TypeScript.

1. **Aim:** To implement inheritance and explore the use of access modifiers in TypeScript.
2. **Objectives:** Aim of this experiment is that, the students will be able
   - To understand example of working with classes in TypeScript and to know basic inheritance features
   - To gain intuition not only for TypeScript's object-oriented language features, but for how and why to use them.
   - To Understand the use of access modifiers at the class level
3. **Outcomes:** After study of this experiment, the students will be able
   - To install Typescript.
   - Write code, compile and execute the code to achieve inheritance.
   - Learn how to use the inheritance to reuse the functionality of another class
   - Understand how the access modifiers change the visibility of the properties and methods of a class.
4. **Prerequisite:** Basic knowledge of JavaScript is required, general concept of inheritance
5. **Requirements:** Personal Computer, Windows operating system, VSCode editor, browser, Internet Connection, google doc.
6. **Pre-Experiment Exercise:**
   Brief Theory: Refer shared material
7. **Laboratory Exercise**
   A. **Procedure:**
      a. **Answer the following:**
         - Inheritance vs Composition
         - Complete the following table:

Access Modifier	Accessible within class	Accessible in subclass	Accessible externally via class instance
Public			
Protected			
Private			

b. **Attach screenshots:**
   - Typescript Program code and output

8. **Post-Experiments Exercise**

A. Extended Theory:

Nil

B. Questions:

- What are Interfaces in TypeScript? (handwritten)
- What are modules in TypeScript? (handwritten)
- open up src/index.ts in your code editor. Enter the following code:

```
let a = 1 + 2
let b = a + 3
let c = {
apple: a,
 banana: b
}
let d = c.apple * 4
```

1) hover over a, b, c, d and notice how and what typescript infers data types of all your variables.
2) Play around the code and see if:

a) Get TypeScript to show a red squiggly when you do something invalid (we call this "throwing a Type Error").

b) Read the Type Error and try to understand what it means.

C. Conclusion:

- Write what was performed in the experiment.
- Write the significance of the topic studied in the experiment.

D. References:

1. https://www.typescriptlang.org/assets/typescript-handbook.pdf
2. http://basarat.gitbooks.io/typescript/
3. Programming TypeScript Making Your JavaScript Applications Scale, by Boris Cherny

Name: Vishal Rajesh Mahajan
Class: TE IT A

Web Lab EXP 03
Roll No: 56

A. Answer the following:

1. Inheritance vs Composition

Inheritance and composition are two fundamental concepts in object-oriented programming used to achieve code reuse.

Feature	Inheritance	Composition
Definition	A class derives (inherits) from another class to acquire its properties and behavior.	A class contains an instance of another class as a field instead of inheriting from it.
Relationship Type	"Is-a" relationship (e.g., Dog **is-a** Animal).	"Has-a" relationship (e.g., Car **has-a** Engine).
Flexibility	Less flexible as changes in the base class affect derived classes.	More flexible as changes in one class don’t affect others directly.
Code Reusability	Code is reuse through inheritance.	Code is reused through delegation.
Encapsulation	Can break encapsulation if a subclass relies on the implementation details of the parent class.	Better encapsulation since internal details are hidden within the composed objects.
Multiple Usage	Supports single and multiple inheritance (in some languages).	Preferred for combining multiple behaviors without the complexity of multiple inheritance.
Example	`class Dog extends Animal { }`	`class Car { private Engine engine; }`

2. Complete the following table:

Access Modifier	Accessible within class	Accessible in subclass	Accessible externally via class instance
Public	Yes	Yes	Yes
Protected	Yes	Yes	No
Private	Yes	No	No

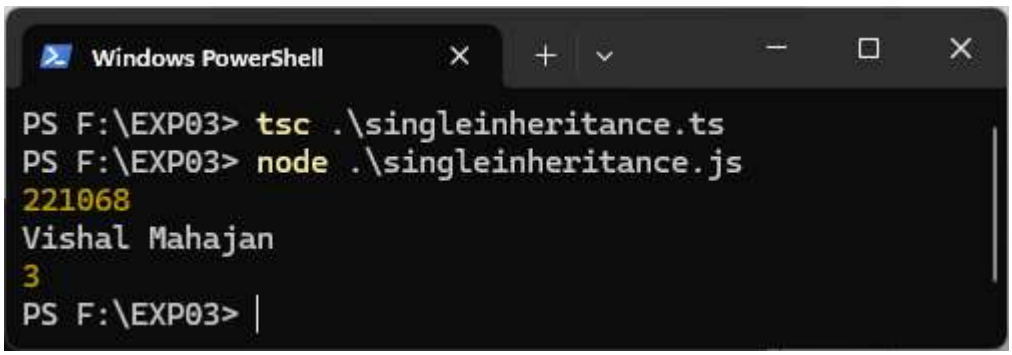
Name: Vishal Rajesh Mahajan
Class: TE IT A

Web Lab EXP 03
Roll No: 56

B. Attach Screenshot

1. Single Inheritance

```
class SFIT {
    PID: number;
    constructor(PID:number){
        this.PID = PID
    }
}
class Student_details extends SFIT{
    stud_name: string;
    year: number;
    constructor(PID:number,stud_name: string, year: number) {
        super(PID);
        this.stud_name = stud_name;
        this.year = year
    }
    display():void{
        console.log(this.PID)
        console.log(this.stud_name)
        console.log(this.year)

    }
}

let SFITobj = new Student_details(221068,"Vishal Mahajan", 3)
SFITobj.display()
```

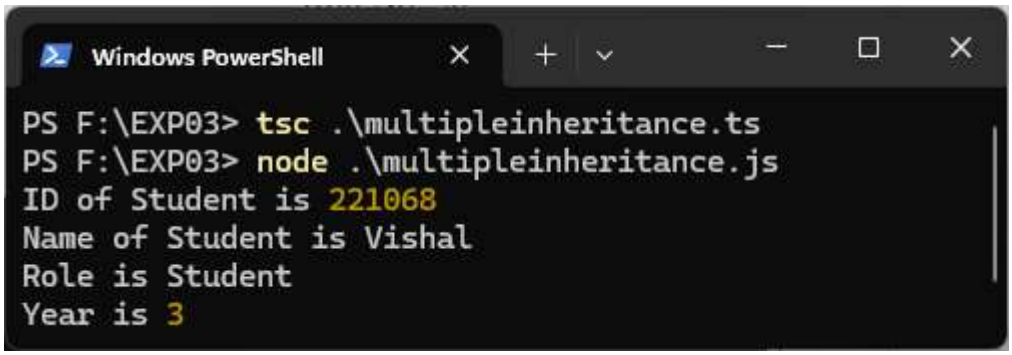


Name: Vishal Rajesh Mahajan
Class: TE IT A

Web Lab EXP 03
Roll No: 56

2. Multiple Inheritance

```ts
interface sfit{
    ID: number
}

interface details{
    name: string
    role: string
}

interface student extends sfit, details{
    year: number
}

let studobj = <student>{}

studobj.ID = 221068
studobj.name = "Vishal"
studobj.role = "Student"
studobj.year = 3

console.log("ID of Student is",studobj.ID)
console.log("Name of Student is", studobj.name)
console.log("Role is", studobj.role)
console.log("Year is",studobj.year)
```



Name: Vishal Rajesh Mahajan
Class: TE IT A

3. Multilevel Inheritance

```
class Sfit {
  PID: number;
  constructor(PID: number) {
    this.PID = PID;
  }
}
class details extends Sfit {
  role: string;
  constructor(PID: number, role: string) {
    super(PID);
    this.role = role;
  }
}
class person extends details {
  name: string;
  constructor(PID: number, role: string, name: string) {
    super(PID, role);
    this.name = name;
  }
  display(): void {
    console.log("PID is", this.PID);
    console.log("Role is", this.role);
    console.log("Name of the Person is", this.name);
  }
}

const multilevel = new person(221068, "Student", "Vishal Mahajan");
multilevel.display();
```

Name: Vishal Rajesh Mahajan
Class: TE IT A

4.Interface

```
interface sfit_college {
  PID: number;
  first_name: string;
  last_name: string;
  year: number;
  GetPID(): number;
  GetName(): string;
  GetYear(): number;
}
class student implements sfit_college {
  PID: number;
  first_name: string;
  last_name: string;
  year: number;
  GetPID(): number {
    return this.PID;
  }
  GetName(): string {
    return this.first_name + " " + this.last_name;
  }
  GetYear(): number {
    return this.year;
  }

  constructor(
    PID: number,
    first_name: string,
    last_name: string,
    year: number
  ) {
    this.PID = PID;
    this.first_name = first_name;
    this.last_name = last_name;
    this.year = year;
  }
}
let student_obj = new student(22106, "Vishal", "Mahajan", 4);
console.log("PID of Student is", student_obj.GetPID());
```

```
console.log("Name of Student is", student_obj.GetName());
console.log("Year of Student is", student_obj.GetYear());
```