

Name of Student	Swaraj Patil
Class Roll No	39
D.O.P.	
D.O.S.	
Sign and Grade	

Title: Understanding Basic Constructs in TypeScript

Aim:

To study basic constructs in TypeScript.

Problem Statement:

1. Create a Base Class Student

- Properties: name, studentId, grade
- Method: getDetails() to display student information.

2. Create a Subclass GraduateStudent

- Extends Student
- Additional property: thesisTopic
- Method: getThesisTopic()
- Override getDetails() to display specific information.

3. Create a Non-Subclass LibraryAccount

- Properties: accountId, booksIssued
- Method: getLibraryInfo()

4. Demonstrate Composition Over Inheritance

- Associate LibraryAccount with Student instead of inheriting from Student.

5. Demonstrate the Implementation

- Create instances of Student, GraduateStudent, and LibraryAccount.
 - Call their methods and observe the behavior of inheritance versus independent class structures.
-

TypeScript Implementation:

1. Student and GraduateStudent Classes

```
class Student {  
    constructor(public name: string, public studentId: number, public grade: string) {}  
  
    getDetails(): string {  
        return `Student Name: ${this.name}, ID: ${this.studentId}, Grade: ${this.grade}`;  
    }  
}  
  
class GraduateStudent extends Student {  
    constructor(name: string, studentId: number, grade: string, public thesisTopic: string) {  
        super(name, studentId, grade);  
    }  
  
    getDetails(): string {  
        return `Graduate Student Name: ${this.name}, ID: ${this.studentId}, Grade: ${this.grade}, Thesis Topic: ${this.thesisTopic}`;  
    }  
  
    getThesisTopic(): string {  
        return `Thesis Topic: ${this.thesisTopic}`;  
    }  
}
```

2. LibraryAccount Class (Independent Class)

```
class LibraryAccount {  
    constructor(public accountId: number, public booksIssued: number) {}  
  
    getLibraryInfo(): string {  
        return `Library Account ID: ${this.accountId}, Books Issued: ${this.booksIssued}`;  
    }  
}
```

```
    }  
  }  
  
3. Demonstrating Composition Over Inheritance  
  
class StudentWithLibrary {  
  constructor(public student: Student, public libraryAccount: LibraryAccount) {}  
  
  getFullDetails(): void {  
    console.log(this.student.getDetails());  
    console.log(this.libraryAccount.getLibraryInfo());  
  }  
}
```

// Creating Instances and Demonstrating Output

```
const gradStudent = new GraduateStudent("Amit", 101, "A", "AI Research");  
const libraryAccount = new LibraryAccount(5001, 3);  
const studentWithLibrary = new StudentWithLibrary(gradStudent, libraryAccount);  
  
studentWithLibrary.getFullDetails();
```

Employee Management System in TypeScript

1. Employee Interface

```
interface Employee {  
  name: string;  
  id: number;  
  role: string;  
  getDetails(): string;  
}
```

2. Manager Class

```
class Manager implements Employee {
```

```
    constructor(public name: string, public id: number, public role: string, public
department: string) { }
```

```
    getDetails(): string {  
        return `Manager Name: ${this.name}, ID: ${this.id}, Role: ${this.role},  
Department: ${this.department}`;  
    }  
}
```

3. Developer Class

```
class Developer implements Employee {
```

```
    constructor(public name: string, public id: number, public role: string, public
programmingLanguages: string[]) { }
```

```
    getDetails(): string {  
        return `Developer Name: ${this.name}, ID: ${this.id}, Role: ${this.role},  
Programming Languages: ${this.programmingLanguages.join(", ")}`;  
    }  
}
```

4. Demonstration of Employee Management System

```
const manager = new Manager("Priya", 201, "Manager", "IT");
```

```
const developer = new Developer("Raj", 202, "Developer", ["TypeScript", "JavaScript"]);
```

```
console.log(manager.getDetails());
```

```
console.log(developer.getDetails());
```

Theory

1. Different Data Types in TypeScript

TypeScript supports several data types, including:

- **Primitive Types:** number, string, boolean
- **Special Types:** any, unknown, never, void
- **Object Types:** array, tuple, enum, interface, class

2. Type Annotations in TypeScript

Type annotations define variable types explicitly. Example:

```
let age: number = 25;
```

```
let name: string = "John";
```

3. Compiling TypeScript Files

TypeScript files (.ts) are compiled to JavaScript using:

```
tsc filename.ts
```

This generates a JavaScript file (filename.js).

4. Difference Between JavaScript and TypeScript

- **JavaScript** is dynamically typed, while **TypeScript** is statically typed.
- TypeScript introduces interfaces, classes, and generics, which JavaScript lacks.
- TypeScript must be compiled into JavaScript before execution.

5. JavaScript vs. TypeScript Inheritance

- **JavaScript:** Uses prototype-based inheritance.
- **TypeScript:** Uses class-based inheritance with extends and implements.

6. Why Use Generics?

Generics provide type safety while making code flexible.

```
function identity<T>(arg: T): T {  
    return arg;  
}  
  
console.log(identity<number>(10));  
console.log(identity<string>("Hello"));
```

Using any loses type safety, while generics retain flexibility without errors.

8. Where Are Interfaces Used?

- Enforcing object structure
- Defining API response structure
- Type checking function parameters
- Implementing contracts in classes

Code:-

```
let age: number = 25;

let personName: string = "John"; // Changed from 'name' to 'personName'

let isStudent: boolean = true;

let scores: number[] = [90, 85, 88];

let user: [string, number] = ["Alice", 30];

enum Direction {

    Up,

    Down,

    Left,

    Right

}

console.log(Direction.Up); // Output: 0 (Default indexing starts from 0)
```

Output

```
[LOG]: "Bhagyesh Patil , 35 "
```

```
[LOG]: 0
```

Code:-

```
let age: number = 25;

let personName: string = "John";

let isStudent: boolean = true;

let scores: number[] = [90, 85, 88];

let fruits: string[] = ["Apple", "Banana", "Mango"];

let user: [string, number] = ["Alice", 30];

enum Direction {

    Up,

    Down,

    Left,

    Right

}

enum Status {

    Active = 1,

    Inactive = 0,

    Pending = -1

}

let randomValue: any = "Hello";

randomValue = 42;

randomValue = true;

let id: number | string;

id = 101;

id = "TS102";

function add(a: number, b: number): number {

    return a + b;

}

function greet(name: string, message?: string): string {

    return `Hello, ${name}! ${message || "Welcome!"}`;

}

interface Person {

    name: string;

    age: number;

    isStudent?: boolean;
```

```
}  
  
let student: Person = {  
    name: "Mark",  
    age: 21,  
    isStudent: true  
};  
  
class Animal {  
    protected name: string;  
  
    constructor(name: string) {  
        this.name = name;  
    }  
  
    public speak(): void {  
        console.log(`${this.name} makes a sound.`);  
    }  
}  
  
class Dog extends Animal {  
    constructor(name: string) {  
        super(name);  
    }  
  
    public speak(): void {  
        console.log(`${this.name} barks!`);  
    }  
}  
  
const myDog = new Dog("Buddy");  
  
function getArray<T>(items: T[]): T[] {  
    return new Array().concat(items);  
}  
  
let numberArray = getArray<number>([1, 2, 3]);  
let stringArray = getArray<string>(["A", "B", "C"]);
```



```
console.log("Age:", age);
console.log("Person Name:", personName);
console.log("Is Student:", isStudent);
console.log("Scores:", scores);
console.log("User Tuple:", user);
console.log("Enum Direction Up:", Direction.Up);
console.log("Enum Status Active:", Status.Active);
console.log("Random Value:", randomValue);
console.log("Union Type ID:", id);
console.log("Addition Result:", add(5, 10));
console.log(greet("John"));
console.log("Student Object:", student);
myDog.speak();
console.log("Generic Number Array:", numberArray);
console.log("Generic String Array:", stringArray);
```

Ouput:-

```
[LOG]: "User Tuple:", ["Bhagyesh", 35]
[LOG]: "Enum Direction Up:", 0
[LOG]: "Enum Status Active:", 1
[LOG]: "Random Value:", true
[LOG]: "Union Type ID:", "TS102"
[LOG]: "Addition Result:", 15
[LOG]: "Hello, Bhagyesh ! Welcome!"
[LOG]: "Student Object:", {
  "name": "Mark",
  "age": 21,
  "isStudent": true
}
[LOG]: "Buddy barks!"
[LOG]: "Generic Number Array:", [1, 2, 3]
[LOG]: "Generic String Array:", ["A", "B", "C"]
```