EXPERIMENT – 1 b: TypeScript

Name of Student	**Sonam Chhabaidiya**
Class Roll No	**D15A_09**
D.O.P.	
D.O.S.	
Sign and Grade	

AIM : To study Basic constructs in TypeScript.

PROBLEM STATEMENT :

a) Create a base class Student with properties like name, studentId, grade, and a method getDetails() to display student information. Create a subclass GraduateStudent that extends Student with additional properties like thesisTopic and a method getThesisTopic(). Override the getDetails() method in GraduateStudent to display specific information. Create a non-subclass LibraryAccount (which does not inherit from Student) with properties like accountId, booksIssued, and a method getLibraryInfo(). Demonstrate composition over inheritance by associating a LibraryAccount object with a Student object instead of inheriting from Student. Create instances of Student, GraduateStudent, and LibraryAccount, call their methods, and observe the behavior of inheritance versus independent class structures.

b) Design an employee management system using TypeScript. Create an Employee interface with properties for name, id, and role, and a method getDetails() that returns employee details. Then, create two classes, Manager and Developer, that implement the Employee interface. The Manager class should include a department property and override the getDetails() method to include the department. The Developer class should include a programmingLanguages array property and override the getDetails() method to include the programming languages. Finally, demonstrate the solution by creating instances of both Manager and Developer classes and displaying their details using the getDetails() method.

THEORY

1. **What are the different data types in TypeScript? What are Type Annotations in Typescript?**
   TypeScript provides several built-in data types:
   - **Primitive Types:** `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`.
   - **Object Types:** `Array`, `Tuple`, `Enum`, `Class`, `Interface`.
   - **Special Types:** `any`, `unknown`, `void`, `never`.

   Type Annotations allow explicit type definitions to prevent errors.
   Example:
   ```
   let age: number = 25;  // Ensures `age` holds only numeric values
   ```

2. **How to Compile TypeScript Files**
   Use the TypeScript compiler (`tsc`) in the terminal:
   ```
   tsc filename.ts
   ```
   This compiles the TypeScript file into JavaScript (`filename.js`).

3. **Difference Between JavaScript and TypeScript**

Feature	JavaScript	TypeScript
Typing	Dynamic	Static (compile-time checking)
Interfaces	Not supported	Supported for structured code
Compilation	Interpreted	Compiled to JavaScript
Inheritance	Prototype-based	Class-based (OOP style)

4. **Inheritance in JavaScript vs. TypeScript**
   - **JavaScript:** Uses **prototype-based inheritance**.
   - **TypeScript:** Uses **class-based inheritance** (`class` and `extends` keywords).

5. **Benefits of Generics in TypeScript**
   - Provides **type safety** while maintaining **flexibility**.
   - Example:
     ```
     function identity<T>(value: T): T { return value; }
     ```
   - Prevents runtime errors, unlike using `any`.

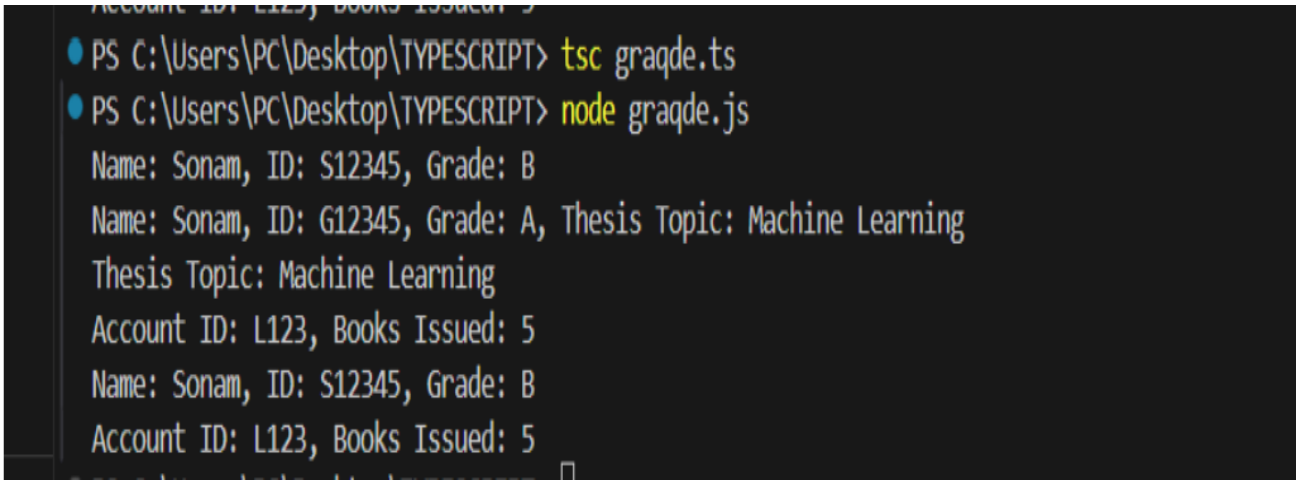
6. Difference Between Classes and Interfaces in TypeScript

Feature	Class	Interface
Definition	Defines properties and methods	Defines structure but no implementation
Instantiation	Can create objects	Cannot be instantiated
Use Case	Used to implement functionality	Used to define object contracts

GITHUB LINK - https://github.com/sonamcc/webx1b

OUTPUT

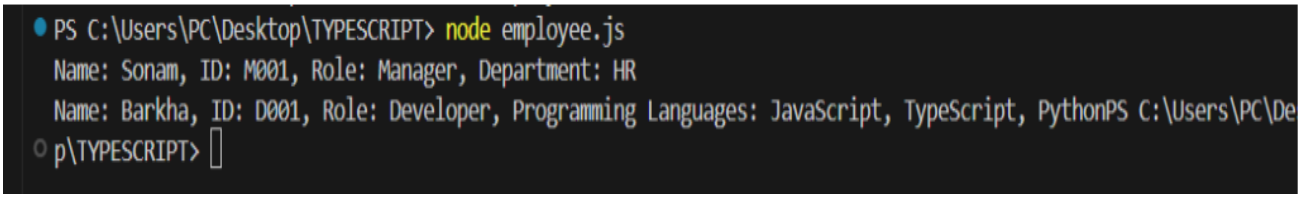
(a) Student and GraduateStudent with Composition



This screenshot displays the output of the TypeScript program implementing **inheritance and composition**. The program first prints details of a **Student** and a **GraduateStudent**,

demonstrating method overriding and inheritance. Then, it prints the **Thesis Topic** of the GraduateStudent separately. The next lines show details of a **LibraryAccount** associated with a student, demonstrating composition. Finally, it displays a combined output of both **Student and LibraryAccount**, showcasing how composition works.

(b) Employee Management System



This screenshot displays the output of the **Employee Management System** program. It shows details of an **Employee interface**, with two classes: **Manager** and **Developer**, implementing it. The output displays the details of a **Manager** named Alex, including their ID, role, and department. It also shows the details of a **Developer** named Anuprita, including their programming languages. The output is generated after running `node src/employee.js` in the terminal.

CONCLUSION

This experiment demonstrated the fundamental concepts of TypeScript, such as inheritance, method overriding, and composition through the implementation of **Student** and **GraduateStudent** classes. Instead of using inheritance, composition was demonstrated by linking **LibraryAccount** with **Student**, emphasizing flexibility in design.

Furthermore, the **Employee Management System** utilized interfaces to enforce structure and type safety, highlighting the advantages of TypeScript in maintaining scalable and well-organized code. Overall, this experiment reinforced the benefits of TypeScript's object-oriented capabilities, improving code readability, reusability, and reliability.