Lecture 10: TypeScript Interfaces and Classes

What is an interface?

- A TypeScript structure that defines the shape of an object

- **Purpose:**
  - Provides type-checking for object structure
  - Enforces consistency across similar objects
  - Helps with code readability and predictability

```
interface Person {
    name: string;
    age: number;
    greet(): string;
}
```

Using Interface with Objects

- Ensures the object matches the interface
- Adds compile-time checks for type safety

```
const person: Person = {
    name: 'Alice',
    age: 25,
    greet() {
        return `Hello, ${this.name}`;
    }
};
```

Extending Interfaces

Allow interfaces to inherit properties from other interfaces without rewriting.

```
interface Employee extends Person {
    employeeId: number;
    department: string;
}
```

Classes in TypeScript

A **blueprint** for creating objects with properties and methods

- Supports encapsualtion, inheritance, and polymorphism
- Includes methods, properties, constructors, and access modifiers (public, private, protected)

Classes in TypeScript

```
class Car {
    make: string;
    model: string;

    constructor(make: string, model: string) {
        this.make = make;
        this.model = model;
    }

    start() {
        return `Starting ${this.make} ${this.model}`;
    }
}
```

Access Modifiers

public

Accessible from anywhere

private

Only accessible within the class

protected

Accessible within the class and subclasses

```
class Animal {
    public name: string;
    protected lastName: string;
    private age: number;

    constructor(name: string,
        lastName: string,
        age: number) {
        this.name = name;
        this.lastName = lastName;
        this.age = age;
    }
}
```

Implementing an Interface in a Class

Enforces class structure and behavior.

```
interface Shape {
    area(): number;
}

class Circle implements Shape {
    radius: number;

    constructor(radius: number) {
        this.radius = radius;
    }

    area(): number {
        return Math.PI * this.radius ** 2;
    }
}
```

Class Inheritance and Extending Classes

Enables a class to extend another class, inheriting its properties and methods

```
class Animal {
    move() {
        console.log("Animal is moving");
    }
}

class Dog extends Animal {
    bark() {
        console.log("Woof!");
    }
}
```

Abstract Classes

A class that cannot be instantiated and is meant to be subclassed

```
abstract class Vehicle {
    abstract fuelType(): string;
    start() {
        console.log("Vehicle is starting");
    }
}

class Car extends Vehicle {
    fuelType() {
        return "Gasoline";
    }
}
```

Interfaces vs Classes

Interfaces

- Used to define the shape of objects
- Do not have implementation details
- Cannot be instantiated

Classes

- Used to create objects with behavior and state
- Can have implementation details, constructors, and modifiers
- Can implement interfaces to enforce structure

When to use Interfaces and Classes

Interfaces

- When defining object shapes, especially for type-checking purposes
- For defining API response structures or configuration objects

Classes

- When creating objects with behavior and logic
- When using inheritance, encapsulation, and polymorphism