Table of Contents

# Bab 4: Classes dan Objects di TypeScript

## Penjelasan Materi

Classes di TypeScript menyediakan cara yang kuat dan terstruktur untuk membuat objek dengan properti dan method. TypeScript memperluas sintaks class JavaScript dengan menambahkan fitur-fitur seperti access modifiers (public, private, protected), abstract classes, interface implementation, dan type checking yang ketat. Ini memungkinkan kita untuk menulis kode berorientasi objek yang lebih aman dan mudah di-maintain.

Classes memungkinkan kita untuk mendefinisikan blueprint untuk objek, menentukan properti dan perilaku yang akan dimiliki oleh setiap instance dari class tersebut. TypeScript menambahkan lapisan keamanan tipe yang memastikan bahwa properti dan method digunakan dengan benar sesuai dengan definisinya.

## Analogi yang Mudah Dipahami

Bayangkan class seperti cetakan kue: - Class adalah cetakan kuenya - Object adalah kue yang dihasilkan - Properties adalah bahan-bahan yang digunakan - Methods adalah langkah-langkah pembuatannya - Constructor adalah resep dasarnya - Inheritance seperti membuat variasi dari resep dasar

Atau bayangkan class seperti blueprint rumah: - Class adalah blueprint/rancangannya - Object adalah rumah yang dibangun - Properties adalah spesifikasi rumah (luas, jumlah kamar, dll) - Methods adalah fungsi-fungsi dalam rumah (buka pintu, nyalakan lampu) - Private members seperti ruangan yang hanya bisa diakses pemilik - Protected members seperti area yang hanya bisa diakses keluarga - Public members seperti area yang bisa diakses siapa saja

## Point Penting

1. **Class Declaration**
   * Sintaks dasar class
   * Constructor
   * Properties dan methods
   * Access modifiers
2. **Inheritance**
   * Extends keyword
   * Super calls
   * Method overriding
   * Abstract classes
3. **Interfaces**
   * Interface implementation
   * Multiple interfaces
   * Optional properties
   * Readonly properties
4. **Access Modifiers**
   * Public (default)
   * Private
   * Protected
   * Parameter properties
5. **Static Members**
   * Static properties
   * Static methods
   * Singleton pattern

## Contoh Kode dan Penjelasan

```typescript // 1. Basic Class class Kendaraan { // Properties dengan access modifiers private \_merek: string; protected \_tahun: number; public warna: string;

// Constructor  
constructor(merek: string, tahun: number, warna: string) {  
 this.\_merek = merek;  
 this.\_tahun = tahun;  
 this.warna = warna;  
}  
  
// Getter  
get merek(): string {  
 return this.\_merek;  
}  
  
// Method  
public getInfo(): string {  
 return \`\${this.\_merek} (\${this.\_tahun}) - \${this.warna}\`;  
}  
  
// Protected method  
protected cekUsia(): number {  
 return new Date().getFullYear() - this.\_tahun;  
}

}

// 2. Inheritance class Mobil extends Kendaraan { private \_jumlahPintu: number;

constructor(  
 merek: string,  
 tahun: number,  
 warna: string,  
 jumlahPintu: number  
) {  
 // Memanggil constructor parent  
 super(merek, tahun, warna);  
 this.\_jumlahPintu = jumlahPintu;  
}  
  
// Override method parent  
public getInfo(): string {  
 return \`\${super.getInfo()} - \${this.\_jumlahPintu} pintu\`;  
}  
  
// Method baru  
public cekLayakJalan(): boolean {  
 return this.cekUsia() < 10;  
}

}

// 3. Interface interface KendaraanElektrik { kapasitasBaterai: number; isiBaterai(jumlah: number): void; }

// 4. Class dengan Interface class MobilListrik extends Mobil implements KendaraanElektrik { public kapasitasBaterai: number; private \_levelBaterai: number;

constructor(  
 merek: string,  
 tahun: number,  
 warna: string,  
 jumlahPintu: number,  
 kapasitasBaterai: number  
) {  
 super(merek, tahun, warna, jumlahPintu);  
 this.kapasitasBaterai = kapasitasBaterai;  
 this.\_levelBaterai = 100;  
}  
  
public isiBaterai(jumlah: number): void {  
 this.\_levelBaterai = Math.min(100, this.\_levelBaterai + jumlah);  
}  
  
public getLevelBaterai(): number {  
 return this.\_levelBaterai;  
}

}

// 5. Abstract Class abstract class Pembayaran { constructor(protected jumlah: number) {}

abstract proses(): boolean;  
  
protected validasiJumlah(): boolean {  
 return this.jumlah > 0;  
}

}

// 6. Concrete Class dari Abstract Class class PembayaranKartuKredit extends Pembayaran { constructor( jumlah: number, private nomorKartu: string ) { super(jumlah); }

public proses(): boolean {  
 if (!this.validasiJumlah()) return false;  
 // Logika pemrosesan kartu kredit  
 console.log(\`Memproses pembayaran \${this.jumlah} dengan kartu \${this.nomorKartu}\`);  
 return true;  
}

}

// 7. Singleton Pattern class Database { private static instance: Database; private constructor() {}

public static getInstance(): Database {  
 if (!Database.instance) {  
 Database.instance = new Database();  
 }  
 return Database.instance;  
}  
  
public query(sql: string): void {  
 console.log(\`Executing query: \${sql}\`);  
}

}

// Penggunaan const mobilSaya = new MobilListrik(“Tesla”, 2023, “Merah”, 4, 85); console.log(mobilSaya.getInfo()); console.log(`Layak Jalan: ${mobilSaya.cekLayakJalan()}`); mobilSaya.isiBaterai(20); console.log(`Level Baterai: ${mobilSaya.getLevelBaterai()}%`);

const pembayaran = new PembayaranKartuKredit(1000000, “1234-5678-9012-3456”); pembayaran.proses();

const db = Database.getInstance(); db.query(“SELECT \* FROM users”); ```

### Penjelasan Setiap Bagian Kode:

1. **Basic Class (Kendaraan)**
   * Mendemonstrasikan properties dengan access modifiers
   * Constructor untuk inisialisasi
   * Getter untuk akses private property
   * Protected method untuk digunakan child classes
2. **Inheritance (Mobil)**
   * Extends untuk inheritance
   * Super call di constructor
   * Method overriding
   * Penggunaan protected method dari parent
3. **Interface (KendaraanElektrik)**
   * Kontrak untuk implementasi
   * Definisi method yang harus diimplementasi
4. **Class dengan Interface (MobilListrik)**
   * Multiple inheritance dengan extends dan implements
   * Implementasi interface method
   * Tambahan properti dan method khusus
5. **Abstract Class (Pembayaran)**
   * Template untuk child classes
   * Abstract method yang harus diimplementasi
   * Protected method yang bisa digunakan child classes
6. **Concrete Class (PembayaranKartuKredit)**
   * Implementasi abstract class
   * Logika bisnis spesifik
   * Penggunaan protected method dari parent
7. **Singleton Pattern (Database)**
   * Private constructor
   * Static instance
   * Static method untuk akses instance

## Cara Kerja TypeScript Classes

1. **Compile-time Checking**:
   * TypeScript memeriksa penggunaan properties dan methods
   * Memastikan access modifiers dipatuhi
   * Validasi implementasi interface
2. **Runtime Behavior**:
   * Dicompile menjadi prototype-based JavaScript
   * Private/protected menjadi closure di JavaScript
   * Type information dihapus saat runtime
3. **Inheritance Chain**:
   * Prototype chain untuk inheritance
   * Method resolution order
   * Constructor chaining

## Tips dan Trik

1. **Parameter Properties Shorthand**

* // ✅ Baik: Menggunakan parameter properties  
  class User {  
   constructor(  
   private name: string,  
   public email: string  
   ) {}  
  }

1. **Method Chaining**

* class Builder {  
   private value: string = "";  
    
   append(str: string): this {  
   this.value += str;  
   return this;  
   }  
    
   getResult(): string {  
   return this.value;  
   }  
  }  
    
  // Penggunaan  
  const result = new Builder()  
   .append("Hello")  
   .append(" ")  
   .append("World")  
   .getResult();

1. **Factory Pattern**

* class Product {  
   static create(type: "A" | "B"): Product {  
   switch (type) {  
   case "A": return new ProductA();  
   case "B": return new ProductB();  
   }  
   }  
  }

## Kesalahan yang Sering Dilakukan Pemula

1. **Mengabaikan Access Modifiers**

* // ❌ Buruk: Semua property public  
  class User {  
   name: string;  
   password: string;  
  }  
    
  // ✅ Baik: Menggunakan access modifiers  
  class User {  
   constructor(  
   public name: string,  
   private password: string  
   ) {}  
  }

1. **Tidak Menggunakan Interface**

* // ❌ Buruk: Class tanpa interface  
  class ApiClient {  
   getData() { /\* ... \*/ }  
   postData() { /\* ... \*/ }  
  }  
    
  // ✅ Baik: Menggunakan interface  
  interface IApiClient {  
   getData(): Promise<any>;  
   postData(data: any): Promise<void>;  
  }  
    
  class ApiClient implements IApiClient {  
   getData() { /\* ... \*/ }  
   postData(data: any) { /\* ... \*/ }  
  }

1. **Inheritance yang Berlebihan**

* // ❌ Buruk: Inheritance chain yang panjang  
  class A extends B extends C extends D { }  
    
  // ✅ Baik: Composition over inheritance  
  class Service {  
   constructor(  
   private logger: Logger,  
   private database: Database  
   ) {}  
  }

### Solusi:

1. Gunakan access modifiers secara tepat
2. Terapkan interface untuk kontrak yang jelas
3. Pilih composition over inheritance
4. Manfaatkan parameter properties
5. Dokumentasikan public API dengan JSDoc