Laporan Praktikum

Mata Kuliah Pemograman Berorientasi Objek

Pertemuan 5 Tugas ke 4

“polymorphysm”

Dosen Pengampu :

Willdan Aprizal Arifin, S.Pd., M.Kom

Disusun Oleh :

Muhammad Zikri Alhaq (2308504)

PROGRAM STUDI SISTEM INFORMASI KELAUTAN

UNIVERSITAS PENDIDIKAN INDONESIA

2024

Tujuan

Memahami dan mengimplementasikan konsep **polymorphism** menggunakan **class** pada JavaScript dengan membuat superclass dan subclass yang menerapkan metode yang sama namun menghasilkan output berbeda.

Dasar Teori

- **Polimorfisme:** Konsep dalam pemrograman berorientasi objek (OOP) yang memungkinkan objek yang berbeda merespons metode yang sama dengan perilaku yang berbeda.
- **Pewarisan (*Inheritance*):** Konsep OOP di mana *subclass* dapat mewarisi properti dan metode dari *superclass*.
- **Kelas dan Objek:** Kelas adalah cetak biru untuk membuat objek, sedangkan objek adalah instance dari kelas tersebut.

Alat dan Bahan

- Laptop atau komputer dengan sistem operasi yang mendukung.
- Browser atau editor teks (seperti VS Code) yang mendukung JavaScript.
- Node.js (opsional untuk menjalankan JavaScript di luar browser).

Langkah-Langkah Percobaan

- **Pembuatan *Superclass* dan *Subclass*:** Kita membuat kelas `Kapal` sebagai *superclass* yang digunakan sebagai dasar. *Subclass* seperti `KapalPenumpang`, `KapalKargo`, `KapalTanker`, `KapalSelam`, dan `KapalPesiar` diwariskan dari kelas ini. Setiap *subclass* dapat meng-override metode `hitungBiayaOperasional()` untuk menampilkan perilaku yang berbeda, yang menunjukkan konsep polimorfisme.

- **Konstruktor *Superclass* dan *Subclass*:** Kelas `Kapal` memiliki konstruktor yang menerima parameter `nama`, `jenis`, `kapasitas`, `panjang`, dan `lebar`. Pada *subclass*, kita menggunakan `super()` untuk memanggil konstruktor *superclass* dan menambahkan properti spesifik masing-masing kelas.

- **Penerapan Polimorfisme:** Pada setiap *subclass*, metode `hitungBiayaOperasional()` di-override untuk menyesuaikan dengan kebutuhan spesifik jenis kapal. Metode yang sama dapat dipanggil pada objek yang berbeda namun memberikan hasil yang berbeda sesuai dengan kelas turunan.

```
class Kapal {
   #nama;
   #jenis;
   #kapasitas;
   #panjang;
   #lebar;

   constructor(nama, jenis, kapasitas, panjang, lebar) {
      this.#nama = nama;
      this.#jenis = jenis;
      this.#kapasitas = kapasitas;
      this.#panjang = panjang;
      this.#lebar = lebar;
   }
```

```
    infoKapal() {
        return `Kapal ${this.#nama} (${this.#jenis}) - Kapasitas: ${this.#kapasitas}, Ukuran:
${this.#panjang}x${this.#lebar}`;
    }

    hitungLuasKapal() {
        return this.#panjang * this.#lebar;
    }

    hitungBiayaOperasional() {
        return this.hitungLuasKapal() * 100;
    }
}

class KapalPenumpang extends Kapal {
    #jumlahDek;

    constructor(nama, kapasitas, panjang, lebar, jumlahDek) {
        super(nama, "Penumpang", kapasitas, panjang, lebar);
        this.#jumlahDek = jumlahDek;
    }

    hitungBiayaOperasional() {
        return super.hitungBiayaOperasional() + (this.#jumlahDek * 1000);
    }
}

class KapalKargo extends Kapal {
    #kapasitasMuatan;

    constructor(nama, kapasitas, panjang, lebar, kapasitasMuatan) {
        super(nama, "Kargo", kapasitas, panjang, lebar);
        this.#kapasitasMuatan = kapasitasMuatan;
    }

    hitungBiayaOperasional() {
        return super.hitungBiayaOperasional() + (this.#kapasitasMuatan * 0.5);
    }
}

class KapalTanker extends Kapal {
    #kapasitasTangki;

    constructor(nama, kapasitas, panjang, lebar, kapasitasTangki) {
        super(nama, "Tanker", kapasitas, panjang, lebar);
        this.#kapasitasTangki = kapasitasTangki;
    }

    hitungBiayaOperasional() {
        return super.hitungBiayaOperasional() + (this.#kapasitasTangki * 0.8);
    }
}

class KapalSelam extends Kapal {
    #kedalamaMaksimum;
```

```
    constructor(nama, kapasitas, panjang, lebar, kedalamaMaksimum) {
        super(nama, "Selam", kapasitas, panjang, lebar);
        this.#kedalamaMaksimum = kedalamaMaksimum;
    }

    hitungBiayaOperasional() {
        return super.hitungBiayaOperasional() + (this.#kedalamaMaksimum * 100);
    }
}

class KapalPesiar extends KapalPenumpang {
    #jumlahFasilitas;

    constructor(nama, kapasitas, panjang, lebar, jumlahDek, jumlahFasilitas) {
        super(nama, kapasitas, panjang, lebar, jumlahDek);
        this.#jumlahFasilitas = jumlahFasilitas;
    }

    hitungBiayaOperasional() {
        return super.hitungBiayaOperasional() + (this.#jumlahFasilitas * 5000);
    }
}
```

Penjelasan Kode

- **Kelas `Kapal`:** Kelas dasar yang berfungsi sebagai template untuk jenis-jenis kapal lain. Metode `hitungBiayaOperasional()` menghitung biaya operasional dasar berdasarkan luas kapal.
- **Kelas `KapalPenumpang`:** Meng-override metode `hitungBiayaOperasional()` dengan menambahkan biaya tambahan berdasarkan jumlah dek.
- **Kelas `KapalKargo`:** Menghitung biaya berdasarkan kapasitas muatan.
- **Kelas `KapalTanker`:** Menambahkan biaya berdasarkan kapasitas tangki.
- **Kelas `KapalSelam`:** Biaya operasional didasarkan pada kedalaman maksimum kapal selam.
- **Kelas `KapalPesiar`:** Biaya ditambah berdasarkan jumlah fasilitas di kapal pesiar.

Kesimpulan

Polimorfisme memungkinkan metode yang sama dipanggil pada objek yang berbeda, namun dengan hasil yang sesuai dengan kebutuhan spesifik dari setiap kelas turunan. Ini memberikan fleksibilitas pada desain kode dan memudahkan pengelolaan berbagai jenis kapal dengan metode yang konsisten.