Laporan Praktikum

Mata Kuliah Pemrograman Berorientasi Objek

Pertemuan 5
“polymorphism”

Dosen Pengampu :

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

Disusun Oleh :

Tegar Dzaki Hakim (2305171)

PROGRAM STUDI SISTEM INFORMASI KELAUTAN

UNIVERSITAS PENDIDIKAN INDONESIA

2024

Tujuan

Tujuan dari praktikum ini adalah untuk memahami dan menerapkan konsep **polimorfisme** pada pemrograman berbasis objek (Object-Oriented Programming atau OOP) menggunakan bahasa JavaScript. Dengan memanfaatkan polimorfisme, kita dapat membuat satu fungsi yang dapat berinteraksi dengan objek dari berbagai subclass dengan metode yang sama, namun menghasilkan keluaran yang berbeda.

Dasar Teori

Polimorfisme adalah salah satu pilar penting dalam OOP yang memungkinkan objek dari berbagai subclass untuk merespons metode yang sama dengan cara yang berbeda. Dalam JavaScript, polimorfisme memungkinkan kita untuk menulis metode di kelas induk, lalu mengubah implementasi metode tersebut pada subclass sesuai dengan kebutuhan.

Dalam konsep inheritance (pewarisan), subclass dapat mewarisi atribut dan metode dari superclass (kelas induk), dan subclass ini dapat memodifikasi atau menambah perilaku baru. Polimorfisme mendorong penggunaan metode yang sama pada objek dari berbagai tipe, sehingga kode menjadi lebih modular, fleksibel, dan mudah dikelola.

Alat dan Bahan

1. Laptop atau komputer dengan sistem operasi yang mendukung pemrograman JavaScript.
2. Text editor atau Integrated Development Environment (IDE) seperti Visual Studio Code, Sublime Text, atau Notepad++.
3. Browser web untuk menjalankan dan menampilkan hasil program (seperti Chrome, Firefox, atau Edge).
4. Node.js (opsional) jika ingin menjalankan JavaScript di luar browser.

Penjelasan Program

Program ini dirancang untuk menunjukkan bagaimana polimorfisme dapat diterapkan dalam konsep pemrograman berbasis objek (OOP) di JavaScript. Pada dasarnya, polimorfisme memungkinkan kita untuk mendefinisikan satu metode di dalam kelas induk (superclass) dan membiarkan subclass mengimplementasikan versi metode tersebut dengan caranya sendiri, sambil tetap menggunakan nama metode yang sama. Ini memberikan fleksibilitas, karena kita dapat berinteraksi dengan objek dari berbagai subclass menggunakan metode yang sama, tanpa harus mengetahui secara detail jenis subclass yang spesifik.

Dalam program ini, terdapat satu kelas induk `Kapal` yang memiliki atribut dan metode dasar untuk mendefinisikan sebuah kapal. Lalu, terdapat lima subclass yang mewakili berbagai jenis kapal, yaitu `KapalPenumpang`, `KapalBarang`, `KapalTanker`, `KapalPesiar`, dan `KapalKontainer`. Masing-masing subclass mewarisi atribut dan metode dari kelas induk `Kapal`, namun mereka mengimplementasikan versi khusus dari metode `infoKapal()` yang disesuaikan dengan jenis kapal yang diwakili oleh subclass tersebut.

1. Kelas Utama: `Kapal`

Kelas `Kapal` adalah dasar dari hierarki pewarisan ini. Kelas ini menyimpan beberapa atribut dasar yang umum dimiliki oleh semua jenis kapal. Atribut-atribut tersebut adalah:

- `#nama`: Menyimpan nama kapal.
- `#jenis`: Menyimpan jenis kapal (misalnya penumpang, barang, tanker, pesiar, atau kontainer).
- `#kapasitas`: Menyimpan kapasitas muatan kapal.
- `#tujuan`: Menyimpan tujuan perjalanan kapal.
- `#_status`: Atribut ini secara default bernilai `'tersedia'` dan menunjukkan status ketersediaan kapal.

Konstruktor pada kelas `Kapal` menerima empat parameter untuk menginisialisasi atribut-atribut `#nama`, `#jenis`, `#kapasitas`, dan `#tujuan` ketika sebuah objek kapal dibuat. Sedangkan atribut `#_status` diinisialisasi secara default sebagai `'tersedia'`.

Kelas `Kapal` juga memiliki beberapa metode dasar, yaitu:

- `getNama()`: Mengembalikan nilai dari atribut `#nama`.
- `getJenis()`: Mengembalikan nilai dari atribut `#jenis`.
- `getKapasitas()`: Mengembalikan nilai dari atribut `#kapasitas`.
- `getTujuan()`: Mengembalikan nilai dari atribut `#tujuan`.
- `infoKapal()`: Metode ini akan menampilkan informasi umum tentang kapal, termasuk nama kapal, jenis kapal, kapasitas, dan tujuan. Ini adalah metode yang akan di-overriden (diubah implementasinya) di subclass.

Sebagai metode dasar, `infoKapal()` bertanggung jawab untuk menampilkan informasi umum tentang kapal. Ini memberikan deskripsi dasar tentang kapal tersebut dan digunakan sebagai pondasi yang dapat diperluas oleh subclass-subclass yang mewarisinya.

2. Subclass: `KapalPenumpang`

Kelas `KapalPenumpang` adalah subclass pertama yang mewarisi dari kelas `Kapal`. Dalam subclass ini, ada properti tambahan yaitu `#tahunProduksi`, yang menyimpan informasi tentang tahun pembuatan kapal penumpang.

Metode `infoKapal()` di-overriden di sini untuk menambahkan informasi tentang tahun produksi kapal, selain informasi umum yang sudah ada di kelas induk. Metode ini memanggil `super.infoKapal()` yang digunakan untuk mengambil informasi dari metode `infoKapal()` milik kelas induk, lalu menambahkan informasi tahun produksi.

Konstruktor dari kelas `KapalPenumpang` mengambil parameter tambahan `tahunProduksi` dan menggunakan `super()` untuk memanggil konstruktor dari kelas induk dan menginisialisasi atribut-atribut yang diwarisi seperti `#nama`, `#kapasitas`, dan `#tujuan`.

3. Subclass: `KapalBarang`

Kapal barang, atau kapal yang khusus digunakan untuk mengangkut barang, diwakili oleh subclass `KapalBarang`. Selain mewarisi atribut dan metode dari kelas `Kapal`, subclass ini menambahkan atribut khusus yaitu `#beratMaksimal`, yang menyimpan informasi tentang berat maksimal barang yang bisa diangkut oleh kapal.

Metode `infoKapal()` di-overriden untuk menampilkan informasi tambahan tentang berat maksimal kapal barang. Sama seperti pada subclass `KapalPenumpang`, metode ini menggunakan

`super.infoKapal()` untuk mendapatkan informasi umum dari kelas induk, lalu menambahkan informasi tambahan khusus untuk kapal barang.

4. Subclass: `KapalTanker`

Kapal tanker adalah kapal yang digunakan untuk mengangkut cairan seperti minyak, gas, atau bahan kimia. Subclass `KapalTanker` menambahkan atribut baru, yaitu `#muatanCairan`, yang menyimpan kapasitas muatan cairan dalam liter.

Metode `infoKapal()` di subclass ini juga di-overriden untuk menampilkan informasi tentang muatan cairan yang dapat dibawa oleh kapal. Seperti subclass lainnya, `super.infoKapal()` dipanggil untuk menampilkan informasi dasar, dan informasi spesifik terkait kapal tanker ditambahkan.

5. Subclass: `KapalPesiar`

Kapal pesiar digunakan untuk keperluan rekreasi, sering kali mengangkut banyak penumpang dalam perjalanan wisata. Subclass `KapalPesiar` menambahkan atribut `#jumlahDek` yang menyimpan jumlah dek yang ada di kapal pesiar.

Metode `infoKapal()` menampilkan informasi tambahan tentang jumlah dek yang dimiliki kapal pesiar, di samping informasi umum yang diwarisi dari kelas induk. Seperti pada subclass lainnya, metode ini menggunakan `super.infoKapal()` untuk mengambil informasi dasar dari kelas induk.

6. Subclass: `KapalKontainer`

Subclass terakhir adalah `KapalKontainer`, yang mewakili kapal yang digunakan untuk mengangkut kontainer dalam satuan TEU (Twenty-foot Equivalent Unit). Subclass ini menambahkan atribut `#kapasitasKontainer` yang menyimpan kapasitas kapal dalam satuan TEU.

Metode `infoKapal()` di subclass ini menambahkan informasi tentang kapasitas kontainer yang bisa diangkut oleh kapal, sambil tetap menggunakan `super.infoKapal()` untuk menampilkan informasi umum kapal.

7. Fungsi ***`tampilkanInfoKapal(kapal)`***

Fungsi ini adalah contoh sederhana penerapan **polimorfisme** dalam program. Fungsi ini hanya menerima satu parameter `kapal`, yang bisa berupa objek dari kelas `Kapal` atau subclass manapun yang mewarisinya. Fungsi ini memanggil metode `infoKapal()` dari objek kapal yang diteruskan, dan karena metode tersebut di-overriden di setiap subclass, hasil yang ditampilkan akan berbeda tergantung pada jenis subclass yang digunakan.

Berikut adalah contoh pemanggilan fungsi `tampilkanInfoKapal()` dengan beberapa objek dari subclass yang berbeda:

- `kapalPenumpang1:` Menampilkan informasi kapal penumpang beserta tahun produksinya.
- `kapalBarang1:` Menampilkan informasi kapal barang dengan berat maksimal muatannya.
- `kapalTanker1:` Menampilkan informasi kapal tanker dengan kapasitas muatan cairan.
- `kapalPesiar1:` Menampilkan informasi kapal pesiar dengan jumlah dek.

- `kapalKontainer1:` Menampilkan informasi kapal kontainer dengan kapasitas kontainernya.

Meskipun fungsi yang dipanggil sama, yaitu `tampilkanInfoKapal()`, keluaran yang dihasilkan berbeda untuk setiap objek. Inilah esensi dari polimorfisme, yaitu penggunaan satu metode yang sama untuk berbagai tipe objek, dengan hasil yang disesuaikan.

Contoh Output

Ketika program ini dijalankan, berikut adalah contoh keluaran yang dihasilkan:

```
Kapal ini Kapal Penumpang Ferry berjenis kapal Penumpang dengan
kapasitas 200 dan tujuan Jakarta dengan tahun produksi 2015
Kapal ini Kapal Kargo berjenis kapal Barang dengan kapasitas 300 dan
tujuan Surabaya dengan berat maksimal 8000 ton
Kapal ini Kapal Tanker Minyak berjenis kapal Tanker dengan kapasitas
500 dan tujuan Balikpapan dengan muatan cairan 2000000 liter
Kapal ini Kapal Pesiar Mewah berjenis kapal Pesiar dengan kapasitas
1000 dan tujuan Bali dengan jumlah dek 10
Kapal ini Kapal Kontainer Raksasa berjenis kapal Kontainer dengan
kapasitas 1000 dan tujuan Singapura dengan kapasitas kontainer 20000
TEU
```

Setiap kapal memberikan detail yang berbeda meskipun semuanya menggunakan metode `infoKapal()` yang sama

Kesimpulan

Melalui praktikum ini, kita berhasil menerapkan konsep polimorfisme dalam pemrograman berbasis objek. Dengan menggunakan polimorfisme, metode yang sama (infoKapal()) dapat diimplementasikan secara berbeda untuk berbagai subclass, sehingga memberikan keluaran yang spesifik untuk setiap jenis kapal. Ini memperlihatkan manfaat OOP dalam membuat kode yang lebih terstruktur, mudah diubah, dan dipelihara.

Polimorfisme memungkinkan kita untuk memanggil metode yang sama pada objek yang berbeda secara konsisten, memberikan fleksibilitas dalam penulisan program dan membuatnya lebih efisien serta mudah untuk diperluas di masa depan.