

Laporan Praktikum
Mata Kuliah Pemrograman Berorientasi Objek



Pertemuan 5 Tugas 4
“Polymorphism”

Dosen Pengampu :
Willdan Aprizal Arifin, S.Pd., M.Kom.

Disusun Oleh :

Hilwa Nur Falah

(2312509)

PROGRAM STUDI SISTEM INFORMASI KELAUTAN
UNIVERSITAS PENDIDIKAN INDONESIA
2024

I. PENDAHULUAN

Pemrograman Berorientasi Objek (PBO) adalah paradigma pemrograman yang menggunakan konsep objek dan kelas. Salah satu konsep penting dalam PBO adalah **polymorphism**, yang memungkinkan objek dari subclass yang berbeda dapat digunakan dengan cara yang sama melalui antarmuka yang seragam. Dalam implementasi JavaScript, polymorphism memfasilitasi penggunaan objek yang berasal dari kelas berbeda untuk memanggil method yang memiliki nama sama tetapi memberikan perilaku yang berbeda sesuai dengan subclass tersebut.

Pada praktikum ini, kita akan mengimplementasikan konsep polymorphism menggunakan class dan subclass dalam JavaScript, di mana setiap subclass memiliki perilaku yang sedikit berbeda meskipun memanggil method yang sama dari superclass. Hal ini bermanfaat untuk menyederhanakan kode dan membuatnya lebih fleksibel dalam menangani berbagai jenis objek yang memiliki karakteristik berbeda.

II. TUJUAN

- a. Memahami konsep **inheritance** (pewarisan) dan **polymorphism** dalam pemrograman berorientasi objek.
- b. Mengimplementasikan subclass yang mewarisi properti dan method dari superclass di JavaScript.
- c. Mempelajari bagaimana polymorphism dapat diterapkan untuk membuat method yang sama memberikan perilaku berbeda pada subclass.
- d. Mampu membuat kode yang dapat memanipulasi objek dengan perilaku yang berbeda secara efisien dan terorganisir.

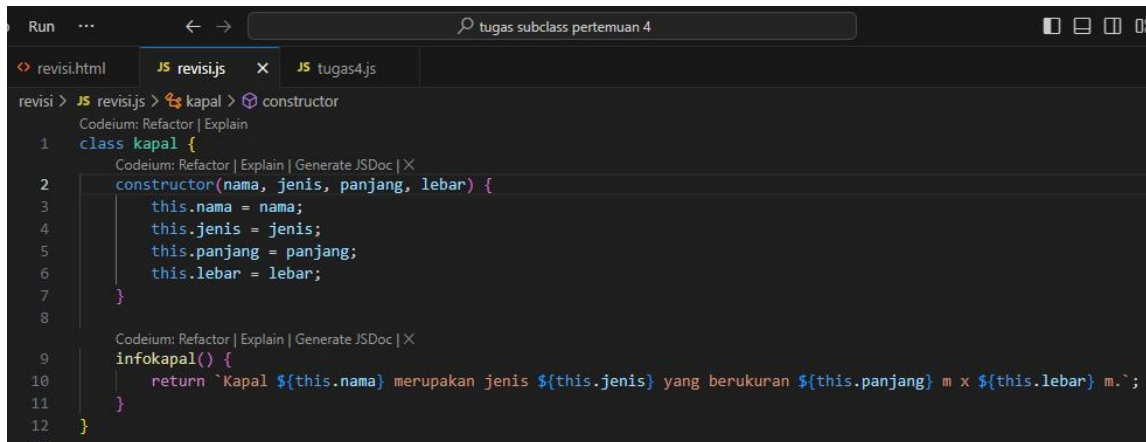
III. ALAT DAN BAHAN

1. Visual Studio Code
2. Google Chrome untuk eksekusi program.

IV. LANGKAH KERJA

1. Membuat Kelas Utama (Kapal)

- Buatlah kelas utama bernama *Kapal*. Kelas ini akan menjadi superclass yang memiliki properti umum, yaitu *nama*, *jenis*, *panjang*, dan *lebar*.
- Tambahkan method *infoKapal()* yang mengembalikan informasi dasar tentang kapal, yaitu nama, jenis, serta ukurannya dalam meter.



```
Run ... tugas subclass pertemuan 4
revisi.html JS revisijs JS tugas4.js
revisi > JS revisijs > kapal > constructor
Codeium: Refactor | Explain
1 class kapal {
  Codeium: Refactor | Explain | Generate JSDoc | X
2   constructor(nama, jenis, panjang, lebar) {
3     this.nama = nama;
4     this.jenis = jenis;
5     this.panjang = panjang;
6     this.lebar = lebar;
7   }
8
9   Codeium: Refactor | Explain | Generate JSDoc | X
10  infoKapal() {
11    return `Kapal ${this.nama} merupakan jenis ${this.jenis} yang berukuran ${this.panjang} m x ${this.lebar} m.`;
12  }
13 }
```

2. Membuat Subclass yang Menerapkan Polymorphism

- KapalCepat*: Subclass ini mewarisi dari *Kapal* dan menambahkan properti *kecepatan*. Method *infoKapal()* ditimpa untuk menambahkan informasi tentang kecepatan kapal.



```
15 // Subclass KapalCepat
Codeium: Refactor | Explain
16 class KapalCepat extends Kapal {
  Codeium: Refactor | Explain | Generate JSDoc | X
17   constructor(nama, jenis, panjang, lebar, kecepatan) {
18     super(nama, jenis, panjang, lebar);
19     this.kecepatan = kecepatan;
20   }
21
  Codeium: Refactor | Explain | Generate JSDoc | X
22   infoKapal() {
23     return `${super.infoKapal()} Kapal ini dapat melaju hingga ${this.kecepatan} knot.`;
24   }
25 }
```

- b. *KapalSelam*: Subclass ini menambahkan properti *kedalamanMaks*, dan method *infoKapal()* ditimpa untuk menambahkan informasi tentang kedalaman penyelaman maksimum kapal.

```
26
27 // Subclass KapalSelam
Codeium: Refactor | Explain
28 class KapalSelam extends Kapal {
Codeium: Refactor | Explain | Generate JSDoc | X
29     constructor(nama, jenis, panjang, lebar, kedalamanMaks) {
30         super(nama, jenis, panjang, lebar);
31         this.kedalamanMaks = kedalamanMaks;
32     }
33
Codeium: Refactor | Explain | Generate JSDoc | X
34     infoKapal() {
35         return `${super.infoKapal()} Kapal ini dapat menyelam hingga kedalaman ${this.kedalamanMaks} meter.`;
36     }
37 }
38
```

- c. *KapalKontainer*: Subclass ini menambahkan properti *kapasitas*, yaitu kapasitas muatan kapal kontainer dalam satuan TEU (Twenty-foot Equivalent Unit).

```
38
39 // Subclass KapalKontainer
Codeium: Refactor | Explain
40 class KapalKontainer extends Kapal {
Codeium: Refactor | Explain | Generate JSDoc | X
41     constructor(nama, jenis, panjang, lebar, kapasitas) {
42         super(nama, jenis, panjang, lebar);
43         this.kapasitas = kapasitas;
44     }
45
Codeium: Refactor | Explain | Generate JSDoc | X
46     infoKapal() {
47         return `${super.infoKapal()} Kapal ini dapat membawa ${this.kapasitas} TEU (Twenty-foot Equivalent Unit) kontaine
48     }
49 }
50
```

- d. *KapalPerang*: Subclass ini menambahkan properti *senjata* dan memperbarui method *infoKapal()* untuk menampilkan jenis senjata yang dimiliki.

```
50
51 // Subclass KapalPerang
52 class KapalPerang extends Kapal {
53     constructor(nama, jenis, panjang, lebar, senjata) {
54         super(nama, jenis, panjang, lebar);
55         this.senjata = senjata;
56     }
57
58     infoKapal() {
59         return `${super.infoKapal()} Kapal ini dilengkapi dengan senjata ${this.senjata}.`;
60     }
61 }
```

- e. *KapalPenangkapIkan*: Subclass ini menambahkan properti *kapasitasTangkap* untuk menyimpan kapasitas penangkapan ikan kapal.

```
62
63 // Subclass KapalPenangkapIkan
64 class KapalPenangkapIkan extends Kapal {
65     constructor(nama, jenis, panjang, lebar, kapasitasTangkap) {
66         super(nama, jenis, panjang, lebar);
67         this.kapasitasTangkap = kapasitasTangkap;
68     }
69
70     infoKapal() {
71         return `${super.infoKapal()} Kapal ini dapat menangkap ikan hingga ${this.kapasitasTangkap} ton.`;
72     }
73 }
74
```

3. Membuat Instance dari Subclass

Buatlah beberapa instance dari subclass yang telah didefinisikan, seperti *KapalCepat*, *KapalSelam*, *KapalKontainer*, *KapalPerang*, dan *KapalPenangkapIkan*. Di sini saya menggunakan *const* karena memastikan bahwa referensi objek tersebut tidak akan diubah secara tidak sengaja di bagian lain dari kode.

```
75 const KapalCepat1 = new KapalCepat("Speedy", "kapal cepat", 100, 30, 50);
76 const KapalSelam1 = new KapalSelam("Deep Explorer", "kapal selam", 80, 20, 300);
77 const KapalKontainer1 = new KapalKontainer("CargoMaster", "kapal kontainer", 350, 60, 10000);
78 const KapalPerang1 = new KapalPerang("Destroyer", "kapal perang", 150, 40, "rudal dan torpedo");
79 const KapalPenangkapIkan1 = new KapalPenangkapIkan("Fishing Pro", "kapal penangkap ikan", 70, 25, 200);
80
```

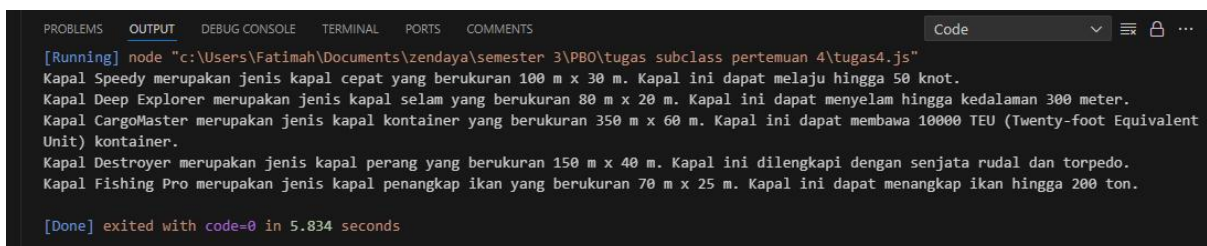
4. Polymorphism dengan *forEach*

Buat array yang berisi semua instance kapal, dan gunakan *forEach()* untuk memanggil method *infoKapal()* dari setiap objek.

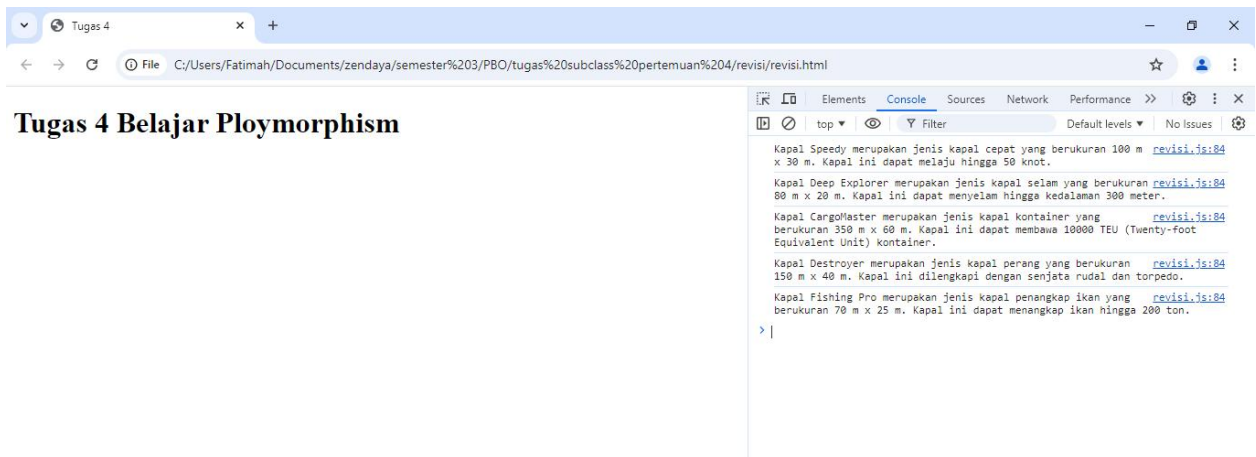
```
80
81 const kapalList = [KapalCepat1, KapalSelam1, KapalKontainer1, KapalPerang1, KapalPenangkapIkan1];
82
83 kapalList.forEach(kapal => {
84     console.log(kapal.infoKapal());
85 });
86
```

5. Eksekusi Kode

Jalankan kode tersebut dan lihat hasil output yang memperlihatkan informasi dari setiap jenis kapal yang ditampilkan berdasarkan subclass yang berbeda.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS  Code
[Running] node "c:\Users\Fatimah\Documents\zendaya\semester 3\PBO\tugas subclass pertemuan 4\tugas4.js"
Kapal Speedy merupakan jenis kapal cepat yang berukuran 100 m x 30 m. Kapal ini dapat melaju hingga 50 knot.
Kapal Deep Explorer merupakan jenis kapal selam yang berukuran 80 m x 20 m. Kapal ini dapat menyelam hingga kedalaman 300 meter.
Kapal CargoMaster merupakan jenis kapal kontainer yang berukuran 350 m x 60 m. Kapal ini dapat membawa 10000 TEU (Twenty-foot Equivalent Unit) kontainer.
Kapal Destroyer merupakan jenis kapal perang yang berukuran 150 m x 40 m. Kapal ini dilengkapi dengan senjata rudal dan torpedo.
Kapal Fishing Pro merupakan jenis kapal penangkap ikan yang berukuran 70 m x 25 m. Kapal ini dapat menangkap ikan hingga 200 ton.
[Done] exited with code=0 in 5.834 seconds
```



V. KESIMPULAN

Dari praktikum ini, dapat disimpulkan bahwa polymorphism memungkinkan objek dari berbagai subclass memiliki method dengan nama yang sama, tetapi dengan implementasi yang berbeda. Dalam contoh ini, method *infoKapal()* pada setiap subclass memberikan output yang berbeda sesuai dengan jenis kapal yang dibuat, meskipun menggunakan nama method yang sama. Ini memudahkan pengelolaan kode, karena setiap instance kapal dapat diperlakukan secara seragam tanpa harus memeriksa jenis subclass secara manual.