

Laporan Modul 6: Inheritance

Mata Kuliah: Praktikum Pemrograman Berorientasi Objek

Nama: Muhammad Alif Arrayyan

NIM: 2024573010085

Kelas: TI-2E

Abstrak

Praktikum ini bertujuan untuk mendalami konsep dasar dan lanjut dari Inheritance (Pewarisan) serta Polymorphism (Polimorfisme) dalam paradigma Pemrograman Berorientasi Objek (PBO) menggunakan bahasa Java. Percobaan dilakukan melalui empat tahapan utama: implementasi Single Inheritance untuk memahami relasi dasar superclass-subclass, eksplorasi Method Overriding dan penggunaan kata kunci super untuk manajemen state objek, penerapan Multilevel dan Hierarchical Inheritance untuk membangun taksonomi kelas yang kompleks, serta integrasi seluruh konsep tersebut dalam sebuah studi kasus Sistem Manajemen Perpustakaan. Hasil praktikum menunjukkan bahwa penerapan mekanisme pewarisan yang tepat dapat meningkatkan efisiensi kode (code reusability), mempermudah pemeliharaan (maintainability), dan memungkinkan perilaku program yang dinamis melalui dynamic binding. Studi kasus akhir mendemonstrasikan bagaimana abstraksi kelas dan polimorfisme memungkinkan pengelolaan berbagai tipe data (Buku, Majalah, DVD) dalam satu struktur koleksi yang terpadu.

Praktikum 1: Memahami Single Inheritance

Dasar Teori

Inheritance adalah konsep fundamental dalam Pemrograman Berorientasi Objek (PBO) yang memungkinkan sebuah class baru (disebut *subclass* atau *child class*) untuk mengadopsi atribut dan method dari class yang sudah ada (disebut *superclass* atau *parent class*). Hal ini mendukung prinsip *Reusability*, di mana kode yang sudah ditulis di superclass tidak perlu ditulis ulang di subclass.

Beberapa konsep kunci dalam praktikum ini adalah:

- Single Inheritance:** Java menerapkan *single inheritance*, artinya satu class hanya boleh mewarisi satu superclass secara langsung menggunakan keyword `extends`.
- Keyword `super`:** Digunakan oleh subclass untuk merujuk pada member (variabel atau method) atau constructor dari superclass. Ini penting untuk memastikan inisialisasi variabel di superclass berjalan dengan benar.
- Access Modifier `protected`:** Memberikan hak akses kepada class itu sendiri, class lain dalam package yang sama, dan subclass (meskipun berada di package berbeda). Ini sering digunakan dalam inheritance agar subclass dapat mengakses atribut superclass secara langsung.
- Method Overriding:** Mekanisme di mana subclass menyediakan implementasi ulang yang spesifik untuk method yang sudah didefinisikan di superclass.

5. **Polymorphism:** Kemampuan objek untuk diperlakukan sebagai bentuk umum (superclass) namun tetap menjalankan perilaku spesifiknya (subclass) saat runtime.

Langkah Praktikum

1. Buat sebuah package baru di dalam package modul_6 dengan nama praktikum_1
2. Buat class Person sebagai superclass:
3. Buat class Student sebagai subclass yang mewarisi Person:
4. Buat class InheritanceTest untuk testing:
5. Jalankan program dan amati hasilnya
6. Perhatikan bagaimana subclass mewarisi dan memperluas fungsionalitas superclass

Hasil Praktikum

Program

```
package modul_6.praktikum_1;

public class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }

    public void greet() {
        System.out.println("Hello, I am a person.");
    }
}

package modul_6.praktikum_1;

public class Student extends Person {
    private String studentId;

    public Student(String name, int age, String studentId) {
        super(name, age); // Memanggil constructor superclass
        this.studentId = studentId;
    }

    public void study() {
        System.out.println(name + " is studying.");
    }

    @Override
    public void greet() {
```

```

        System.out.println("Hello, I am a student named " + name);
    }
}

package modul_6.praktikum_1;

public class InheritanceTest {
    public static void main(String[] args) {
        Student student = new Student("Alice", 20, "S12345");

        // Memanggil method dari superclass
        student.displayInfo();

        // Memanggil method dari subclass
        student.study();

        // Memanggil overridden method
        student.greet();

        // Polymorphism: Student sebagai Person
        Person person = new Student("Bob", 22, "S67890");
        person.greet(); // Memanggil method yang di-override
    }
}

```

Output

```

C:\amazon-corretto-21.0.8.9.1-windows-x64-jdk\j
Name: Alice
Age: 20
Alice is studying.
Hello, I am a student named Alice
Hello, I am a student named Bob

Process finished with exit code 0

```

Analisa dan Pembahasan

Pada praktikum mengenai *Single Inheritance* ini, kita membangun hierarki class sederhana yang terdiri dari superclass `Person` dan subclass `Student`. Berikut adalah analisa mendalam terhadap kode program dan jalannya eksekusi:

1. Implementasi Pewarisan (Inheritance) Class `Student` dideklarasikan dengan `public class Student extends Person`. Keyword `extends` menandakan bahwa `Student` mewarisi seluruh atribut (`name`, `age`) dan method (`displayInfo`, `greet`) yang bersifat *visible* dari class `Person`. Hasilnya, objek `Student` dapat menggunakan method `displayInfo()` meskipun method tersebut tidak ditulis di dalam class `Student`.

2. Konstruktor dan Keyword `super` Pada constructor `Student`, terdapat baris `super(name, age);`.

- Ini berfungsi untuk memanggil constructor dari class `Person`.

- Karena atribut `name` dan `age` dikelola oleh `Person`, tugas inisialisasi nilai diserahkan ke superclass terlebih dahulu sebelum subclass menginisialisasi atribut khususnya sendiri (`studentId`). Jika baris ini dihilangkan atau tidak diletakkan di baris pertama, program akan mengalami error.

3. Hak Akses `protected` Di class `Person`, atribut `name` dan `age` menggunakan modifier `protected`.

- Hal ini memungkinkan class `Student` (sebagai turunan) untuk mengakses variabel `name` secara langsung pada method `study()` (`System.out.println(name + " is studying.");`).
- Jika diganti menjadi `private`, `Student` tidak bisa mengaksesnya langsung dan harus menggunakan method *getter*.

4. Method Overriding Method `greet()` terdapat di kedua class. Di class `Student`, kita melakukan **Overriding** terhadap method `greet()` milik `Person`.

- Saat `student.greet()` dipanggil, program menjalankan versi yang ada di `Student` ("Hello, I am a student...").
- Ini menunjukkan bahwa subclass dapat mengubah perilaku standar yang diwarisi dari superclass sesuai kebutuhan spesifiknya.

5. Pengujian Polimorfisme Pada baris kode `Person person = new Student("Bob", 22, "S67890");`, terjadi konsep polimorfisme.

- Variabel referensi bertipe `Person`, namun objek aslinya adalah `Student`.
- Saat `person.greet()` dipanggil, Java menggunakan *Dynamic Binding* (runtime polymorphism). Mesin Java melihat bahwa objek aslinya adalah `Student`, sehingga method yang dieksekusi tetaplah method `greet()` milik `Student` yang sudah di-override, bukan milik `Person`.

Praktikum 2: Method Overriding dan Kata Kunci super

Dasar Teori

1. Method Overriding Method Overriding adalah fitur dalam konsep pewarisan (inheritance) yang memungkinkan subclass untuk menyediakan implementasi spesifik dari method yang sudah dideklarasikan oleh superclass-nya. Syarat utama overriding adalah:

- Nama method harus sama.
- Daftar parameter (argumen) harus sama.
- Tipe kembalian (return type) harus kompatibel.
- Access modifier tidak boleh lebih ketat daripada method di superclass (misalnya, jika superclass `public`, subclass tidak boleh `protected` atau `private`).

2. Kata Kunci `super` Keyword `super` adalah variabel referensi yang digunakan untuk merujuk pada objek dari superclass langsung. Penggunaannya dalam praktikum ini meliputi:

- **Mengakses Constructor Superclass:** `super(parameter)` digunakan untuk memanggil constructor parent class. Ini harus menjadi pernyataan pertama dalam constructor subclass.
- **Mengakses Method Superclass:** `super.namaMethod()` digunakan untuk memanggil method versi parent class, seringkali dilakukan ketika kita ingin mempertahankan logika asli parent class sambil menambahkan logika baru di subclass.

3. Polymorphism (Dynamic Binding) Dalam overriding, Java menggunakan *dynamic binding* atau *runtime polymorphism*. Artinya, versi method yang dieksekusi ditentukan berdasarkan tipe objek yang sebenarnya dibuat di memori (saat runtime), bukan berdasarkan tipe variabel referensinya.

Langkah Praktikum

1. Buat sebuah package baru di dalam package modul_6 dengan nama praktikum_2
2. Buat class Vehicle sebagai superclass:
3. Buat class Car sebagai subclass yang mewarisi Vehicle:
4. Buat class OverrideTest untuk testing:
5. Jalankan program dan amati:
 - Cara method overriding bekerja
 - Penggunaan kata kunci super untuk mengakses superclass
 - Konsep polymorphism dalam inheritance

Hasil Praktikum

Program

```
package modul_6.praktikum_2;

public class Vehicle {
    protected String brand;
    protected int speed;

    public Vehicle(String brand, int speed) {
        this.brand = brand;
        this.speed = speed;
    }

    public void start() {
        System.out.println("Vehicle is starting...");
    }

    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Speed: " + speed + " km/h");
    }
}

package modul_6.praktikum_2;

public class Car extends Vehicle {
    private int numberOfDoors;

    public Car(String brand, int speed, int numberOfDoors) {
        super(brand, speed); // Memanggil constructor superclass
        this.numberOfDoors = numberOfDoors;
    }

    @Override
    public void start() {
```

```

        super.start(); // Memanggil method start dari superclass
        System.out.println("Car engine is running smoothly");
    }

    @Override
    public void displayInfo() {
        super.displayInfo(); // Memanggil method displayInfo dari superclass
        System.out.println("Number of doors: " + numberOfDoors);
    }

    public void honk() {
        System.out.println("Beep beep!");
    }
}

package modul_6.praktikum_2;

public class OverrideTest {
    public static void main(String[] args) {
        Car car = new Car("Toyota", 180, 4);

        // Memanggil overridden method
        car.start();
        car.displayInfo();
        car.honk();

        // Demonstrasi polymorphism
        Vehicle vehicle = new Car("Honda", 200, 2);
        vehicle.start(); // Memanggil method yang di-override
        vehicle.displayInfo(); // Memanggil method yang di-override
    }
}

```

Output

```

C:\amazon-corretto-21.0.8.9.1-windows-x64-jdk\
Vehicle is starting...
Car engine is running smoothly
Brand: Toyota
Speed: 180 km/h
Number of doors: 4
Beep beep!
Vehicle is starting...
Car engine is running smoothly
Brand: Honda
Speed: 200 km/h
Number of doors: 2

Process finished with exit code 0

```

Analisa dan Pembahasan

Pada program `Vehicle.java`, analisis difokuskan pada strategi enkapsulasi dalam konteks pewarisan. Penggunaan *access modifier* `protected` pada atribut `brand` dan `speed` bukanlah keputusan trivial; hal ini menciptakan *trade-off* antara keamanan data dan fleksibilitas turunan. Jika menggunakan `private`, memori atribut tersebut akan terisolasi total sehingga subclass harus mengaksesnya melalui beban komputasi tambahan (method *getter/setter*). Dengan `protected`, atribut tersebut disimpan dalam area memori yang dapat diakses langsung oleh hierarki pewarisannya (*package-private* + *subclass*), memungkinkan manipulasi *state* objek yang lebih efisien oleh subclass. Class ini bertindak sebagai *blueprint* generik yang mendefinisikan kontrak dasar (method `start` dan `displayInfo`) yang wajib dimiliki oleh entitas kendaraan, namun implementasinya sengaja dibuat umum agar dapat dispesifikasi ulang (*re-defined*) oleh kelas turunannya.

Program `Car.java` mendemonstrasikan kompleksitas siklus hidup objek (*object lifecycle*) melalui mekanisme *Constructor Chaining*. Ketika `new Car(...)` dipanggil, alokasi memori di *Heap* disiapkan untuk menampung seluruh atribut `Vehicle` dan `Car`. Perintah `super(brand, speed)` bukan sekadar pemanggilan method, melainkan instruksi wajib kepada JVM untuk menginisialisasi bagian memori milik superclass terlebih dahulu. Tanpa langkah ini, objek akan berada dalam status *invalid state* karena atribut warisannya bernilai *null* atau 0. Analisis pada method overriding (`start` dan `displayInfo`) menunjukkan penerapan pola *Decorator*, di mana subclass tidak menimpa total logika superclass, melainkan membungkusnya. Saat `super.start()` dieksekusi, kendali program melompat sementara ke blok memori `Vehicle` untuk menjalankan instruksi dasar, lalu kembali ke blok `Car` untuk menjalankan instruksi tambahan. Ini menjamin prinsip *Code Reusability* tetap terjaga.

Program `OverrideTest.java` adalah tempat terjadinya mekanisme *Upcasting* dan *Dynamic Binding*. Pada baris `Vehicle vehicle = new Car(...)`, variabel referensi `vehicle` disimpan di *Stack Memory*, sedangkan objek aslinya `Car` disimpan di *Heap Memory*. Analisis krusial terjadi di sini: secara *Compile-Time*, compiler hanya mengenali referensi `vehicle`, sehingga ia membatasi akses hanya pada method yang didefinisikan di class `Vehicle` (itulah sebabnya `vehicle.honk()` error). Namun, secara *Run-Time*, JVM menggunakan tabel metode virtual (*vtable*) untuk memeriksa tipe objek sebenarnya di *Heap*. Karena objek aslinya adalah `Car`, maka pointer method diarahkan ke implementasi `start()` milik `Car`. Inilah esensi dari polimorfisme: pemisahan antara definisi antarmuka (oleh tipe referensi) dan implementasi perilaku (oleh tipe objek).

Praktikum 3: Multilevel dan Hierarchical Inheritance

Dasar Teori

1. Multilevel Inheritance (Pewarisan Bertingkat) Multilevel Inheritance adalah pola pewarisan di mana sebuah subclass mewarisi superclass, dan kemudian subclass tersebut menjadi superclass bagi class lainnya. Ini membentuk rantai pewarisan linear ($A \rightarrow B \rightarrow C$).

- Dalam konsep ini, class C mewarisi sifat dari B, dan secara transitif juga mewarisi sifat dari A.
- Prinsip utamanya adalah **spesialisasi bertahap**: Class A bersifat umum, B lebih spesifik, dan C sangat spesifik.

2. Hierarchical Inheritance (Pewarisan Hierarki) Hierarchical Inheritance terjadi ketika satu superclass diwarisi oleh beberapa subclass yang berbeda secara bersamaan ($A \rightarrow B$ dan $A \rightarrow C$).

- Ini memungkinkan pembagian kode (*code sharing*) yang efisien untuk sifat-sifat umum.
- Meskipun B dan C memiliki induk yang sama, mereka adalah entitas yang independen dan dapat memiliki implementasi metode yang berbeda (*polimorfisme*).

3. Heterogeneous Collection (Polimorfisme dalam Array) Salah satu kekuatan utama pewarisan adalah kemampuan untuk mengelompokkan objek-objek dari subclass yang berbeda ke dalam satu wadah (array atau list) yang bertipe superclass.

- Contoh: `Animal[]` dapat menyimpan objek `Dog` dan `Cat` .
- Saat dilakukan iterasi pada array tersebut, Java menggunakan mekanisme *Dynamic Binding* untuk menentukan method mana yang dijalankan berdasarkan tipe objek aslinya di memori, bukan tipe array-nya.

Langkah Praktikum

1. Buat sebuah package baru di dalam package `modul_6` dengan nama `praktikum_3`
2. Buat class `Animal` sebagai superclass:
3. Buat class `Mammal` yang mewarisi `Animal` (multilevel inheritance):
4. Buat class `Dog` yang mewarisi `Mammal` (multilevel inheritance):
5. Buat class `Cat` yang mewarisi `Mammal` (hierarchical inheritance):
6. Buat class `InheritanceTypeTest` untuk testing:
7. Jalankan program dan amati:
 - Cara multilevel inheritance membentuk rantai pewarisan.
 - Cara hierarchical inheritance memungkinkan multiple class mewarisi dari satu superclass.
 - Polymorphism bekerja dengan inheritance hierarchy.

Hasil Praktikum

Program

```
package modul_6.praktikum_3;

public class Animal {
    protected String name;

    public Animal(String name) {
        this.name = name;
    }

    public void eat() {
        System.out.println(name + " is eating.");
    }

    public void sleep() {
        System.out.println(name + " is sleeping.");
    }
}
```

```
package modul_6.praktikum_3;

public class Mammal extends Animal {
    protected String furColor;

    public Mammal(String name, String furColor) {
        super(name);
        this.furColor = furColor;
    }
}
```



```

    }

    public void giveBirth() {
        System.out.println(name + " is giving birth to live young.");
    }
}

```

```

package modul_6.praktikum_3;

public class Dog extends Mammal {
    private String breed;

    public Dog(String name, String furColor, String breed) {
        super(name, furColor);
        this.breed = breed;
    }

    public void bark() {
        System.out.println(name + " is barking: Woof woof!");
    }

    @Override
    public void eat() {
        System.out.println(name + " the dog is eating dog food.");
    }
}

```

```

package modul_6.praktikum_3;

public class Cat extends Mammal {
    private boolean isIndoor;

    public Cat(String name, String furColor, boolean isIndoor) {
        super(name, furColor);
        this.isIndoor = isIndoor;
    }

    public void meow() {
        System.out.println(name + " is meowing: Meow meow!");
    }

    @Override
    public void eat() {
        System.out.println(name + " the cat is eating cat food.");
    }
}

```

```

package modul_6.praktikum_3;

public class InheritanceTypeTest {
    public static void main(String[] args) {
        // Multilevel inheritance test
        Dog dog = new Dog("Buddy", "Brown", "Golden Retriever");
        dog.eat(); // Dari Animal, di-override di Dog
        dog.sleep(); // Dari Animal
    }
}

```

```

        dog.giveBirth(); // Dari Mammal
        dog.bark(); // Dari Dog

        System.out.println();

        // Hierarchical inheritance test
        Cat cat = new Cat("Whiskers", "White", true);
        cat.eat(); // Dari Animal, di-override di Cat
        cat.sleep(); // Dari Animal
        cat.giveBirth(); // Dari Mammal
        cat.meow(); // Dari Cat

        System.out.println();

        // Polymorphism dengan hierarchical inheritance
        Animal[] animals = {new Dog("Max", "Black", "Labrador"),
                             new Cat("Luna", "Gray", false)};

        for (Animal animal : animals) {
            animal.eat(); // Akan memanggil method yang sesuai dengan object sebenarnya
        }
    }
}

```

Output

```

C:\amazon-corretto-21.0.8.9.1-windows-x64-jdk\jdk21
Buddy the dog is eating dog food.
Buddy is sleeping.
Buddy is giving birth to live young.
Buddy is barking: Woof woof!

Whiskers the cat is eating cat food.
Whiskers is sleeping.
Whiskers is giving birth to live young.
Whiskers is meowing: Meow meow!

Max the dog is eating dog food.
Luna the cat is eating cat food.

Process finished with exit code 0

```

Analisa dan Pembahasan

Analisa pada praktikum ini meninjau arsitektur pewarisan yang lebih kompleks, menggabungkan *Multilevel* dan *Hierarchical Inheritance* untuk membentuk struktur taksonomi yang realistis.

1. Arsitektur Pewarisan: Transitivitas dan Spesialisasi Program ini membangun struktur hierarki tiga level: `Animal`

→ `Mammal` → `Dog` / `Cat` .

- **Transitivitas Sifat:** Pada class `Dog`, meskipun ia hanya secara eksplisit melakukan `extends Mammal`, ia secara otomatis memiliki akses terhadap method `sleep()` milik `Animal`. Ini membuktikan sifat transitif dalam *Multilevel Inheritance*: jika `Dog` adalah Mamalia, dan Mamalia adalah Hewan, maka `Dog` adalah Hewan. Secara memori, objek `Dog` mengandung lapisan data dari `Dog`, `Mammal`, dan `Animal`.
- **Percabangan Hierarki:** Class `Mammal` bertindak sebagai titik percabangan (*branching point*) bagi `Dog` dan `Cat`. Ini adalah *Hierarchical Inheritance*. Kedua subclass ini berbagi atribut `furColor` dan perilaku `giveBirth()` yang didefinisikan di `Mammal`, namun tetap terisolasi satu sama lain dalam hal perilaku spesifik (`bark` vs `meow`).

2. Manajemen State melalui Constructor Chaining 3 Level Kompleksitas manajemen memori terlihat jelas pada saat inisialisasi objek, misalnya `new Dog("Buddy", ...)`. Terjadi eskalasi pemanggilan konstruktor berantai (*cascading constructor calls*):

1. Konstruktor `Dog` dipanggil → memanggil `super()` ke `Mammal`.
2. Konstruktor `Mammal` dipanggil → memanggil `super()` ke `Animal`.
3. Konstruktor `Animal` dieksekusi untuk mengisi `name`.
4. Alur kembali ke `Mammal` untuk mengisi `furColor`.
5. Alur kembali ke `Dog` untuk mengisi `breed`. **Analisa:** Mekanisme ini menjamin integritas data dari level teratas hingga terbawah. Setiap lapisan kelas bertanggung jawab menginisialisasi atribut yang dimilikinya sendiri, menjaga prinsip *Encapsulation* tetap utuh meskipun dalam rantai pewarisan yang panjang.

3. Polimorfisme dan Virtual Method Table (vtable) Pada class `InheritanceTypeTest`, terdapat implementasi array polimorfik:

```
Animal[] animals = {new Dog(...), new Cat(...)};
for (Animal animal : animals) { animal.eat(); }
```

Praktikum 4: Sistem Manajemen Perpustakaan Sederhana

Dasar Teori

1. Multilevel Inheritance (Pewarisan Bertingkat) *Multilevel Inheritance* adalah jenis pewarisan di mana sebuah class mewarisi class lain, dan class tersebut kemudian diwarisi lagi oleh class berikutnya. Ini membentuk rantai pewarisan linear (misalnya: Class A → Class B → Class C).

- **Prinsip Transitif:** Class C mewarisi sifat dari B, dan karena B mewarisi sifat dari A, maka secara otomatis C juga memiliki akses ke sifat-sifat publik/protected milik A.
- **Hierarki Spesialisasi:** Semakin ke bawah rantai pewarisan, class tersebut menjadi semakin spesifik.

2. Hierarchical Inheritance (Pewarisan Hierarki) *Hierarchical Inheritance* terjadi ketika satu *superclass* (induk) diwarisi oleh lebih dari satu *subclass* (anak) secara langsung. (Misalnya: Class A diwarisi oleh Class B dan Class C).

- **Pembagian Kode (Code Sharing):** Sifat-sifat umum diletakkan di superclass A, sehingga Class B dan C tidak perlu menulis ulang kode yang sama.
- **Independensi Subclass:** Meskipun Class B dan Class C berbagi induk yang sama, mereka tidak memiliki hubungan langsung satu sama lain (independen).

3. Rantai Konstruktor (*Constructor Chaining*) Dalam struktur pewarisan bertingkat, inisialisasi objek melibatkan pemanggilan konstruktor secara berantai dari subclass terbawah hingga superclass teratas.

- Ketika objek class C dibuat, konstruktor C memanggil konstruktor B (menggunakan `super()`), dan konstruktor B memanggil konstruktor A.
- Ini memastikan bahwa setiap lapisan data dalam objek terinisialisasi dengan benar sebelum digunakan.

4. Polimorfisme dalam Koleksi Heterogen Kombinasi *hierarchical inheritance* memungkinkan penggunaan polimorfisme untuk mengelompokkan objek yang berbeda tipe namun memiliki induk yang sama.

- Kita dapat membuat array bertipe Superclass (misalnya `Animal[]`) untuk menyimpan berbagai objek Subclass (seperti `Dog` dan `Cat`).
- Saat metode dipanggil melalui iterasi array, Java menggunakan *Dynamic Binding* untuk mengeksekusi metode yang sesuai dengan tipe asli objek tersebut di memori.

Langkah Praktikum

1. Buat sebuah package baru di dalam package modul_6 dengan nama praktikum_4
2. Buat class `LibraryItem` sebagai superclass:
3. Buat class `Book` yang mewarisi `LibraryItem`:
4. Buat class `Magazine` yang mewarisi `LibraryItem`:
5. Buat class `DVD` yang mewarisi `LibraryItem`:
6. Buat class `LibraryManagementSystem` sebagai main class:
7. Jalankan program dan uji semua fitur:
 - Menampilkan semua item
 - Meminjam dan mengembalikan item
 - Menambah item baru
 - Mencari item berdasarkan kata kunci

Hasil Praktikum

Program

```
package modul_6.praktikum_4;

public abstract class LibraryItem {
    protected String itemId;
    protected String title;
    protected int year;
    protected boolean isAvailable;

    public LibraryItem(String itemId, String title, int year) {
        this.itemId = itemId;
        this.title = title;
        this.year = year;
        this.isAvailable = true;
    }

    // Getter methods
    public String getItemId() { return itemId; }
    public String getTitle() { return title; }
```

```

    public int getYear() { return year; }
    public boolean isAvailable() { return isAvailable; }

    // Setter methods
    public void setAvailable(boolean available) { isAvailable = available; }

    // Abstract method yang harus diimplementasikan subclass
    public abstract void displayInfo();

    // Concrete method yang bisa digunakan semua subclass
    public void borrowItem() {
        if (isAvailable) {
            isAvailable = false;
            System.out.println(title + " berhasil dipinjam");
        } else {
            System.out.println(title + " sedang tidak tersedia");
        }
    }

    public void returnItem() {
        isAvailable = true;
        System.out.println(title + " berhasil dikembalikan");
    }
}

package modul_6.praktikum_4;

public class Book extends LibraryItem {
    private String author;
    private String isbn;
    private int numberOfPages;

    public Book(String itemId, String title, int year, String author, String isbn, int numberOfPages) {
        super(itemId, title, year);
        this.author = author;
        this.isbn = isbn;
        this.numberOfPages = numberOfPages;
    }

    @Override
    public void displayInfo() {
        System.out.println("--- BUKU ---");
        System.out.println("ID: " + itemId);
        System.out.println("Judul: " + title);
        System.out.println("Penulis: " + author);
        System.out.println("Tahun: " + year);
        System.out.println("ISBN: " + isbn);
        System.out.println("Jumlah Halaman: " + numberOfPages);
        System.out.println("Status: " + (isAvailable ? "Tersedia" : "Dipinjam"));
        System.out.println("-----");
    }

    // Method khusus Book
    public void readSample() {
        System.out.println("Membaca sample dari buku: " + title);
    }
}

```

```

package modul_6.praktikum_4;

public class Magazine extends LibraryItem {
    private String publisher;
    private int issueNumber;
    private String category;

    public Magazine(String itemId, String title, int year, String publisher, int issueNumber, String category) {
        super(itemId, title, year);
        this.publisher = publisher;
        this.issueNumber = issueNumber;
        this.category = category;
    }

    @Override
    public void displayInfo() {
        System.out.println("--- MAJALAH ---");
        System.out.println("ID: " + itemId);
        System.out.println("Judul: " + title);
        System.out.println("Penerbit: " + publisher);
        System.out.println("Tahun: " + year);
        System.out.println("Edisi: " + issueNumber);
        System.out.println("Kategori: " + category);
        System.out.println("Status: " + (isAvailable ? "Tersedia" : "Dipinjam"));
        System.out.println("-----");
    }

    // Method khusus Magazine
    public void browseArticles() {
        System.out.println("Menelusuri artikel dalam majalah: " + title);
    }
}

```

```

package modul_6.praktikum_4;

public class DVD extends LibraryItem {
    private String director;
    private int duration; // dalam menit
    private String genre;

    public DVD(String itemId, String title, int year, String director, int duration, String genre) {
        super(itemId, title, year);
        this.director = director;
        this.duration = duration;
        this.genre = genre;
    }

    @Override
    public void displayInfo() {
        System.out.println("--- DVD ---");
        System.out.println("ID: " + itemId);
        System.out.println("Judul: " + title);
        System.out.println("Sutradara: " + director);
        System.out.println("Tahun: " + year);
        System.out.println("Durasi: " + duration + " menit");
        System.out.println("Genre: " + genre);
        System.out.println("Status: " + (isAvailable ? "Tersedia" : "Dipinjam"));
    }
}

```

```

        System.out.println("-----");
    }

    // Method khusus DVD
    public void playTrailer() {
        System.out.println("Memutar trailer DVD: " + title);
    }
}

package modul_6.praktikum_4;

import java.util.ArrayList;
import java.util.Scanner;

public class LibraryManagementSystem {

    // List untuk menyimpan semua item perpustakaan (menggunakan polymorphism)
    private static ArrayList<LibraryItem> libraryItems = new ArrayList<>();

    // Scanner global untuk input pengguna
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

        // Inisialisasi data contoh saat program dimulai
        initializeSampleData();

        System.out.println("Selamat Datang di Sistem Manajemen Perpustakaan!");

        // Loop utama program
        while (true) {
            displayMenu();

            int choice = 0;
            if (scanner.hasNextInt()) {
                choice = scanner.nextInt();
                scanner.nextLine(); // consume newline
            } else {
                System.out.println("Input tidak valid. Harap masukkan angka.");
                scanner.nextLine(); // consume invalid input
                continue;
            }

            switch (choice) {
                case 1:
                    displayAllItems();
                    break;
                case 2:
                    addNewItem();
                    break;
                case 3:
                    borrowItem();
                    break;
                case 4:
                    returnItem();
                    break;
                case 5:
                    searchItem();
                    break;
            }
        }
    }
}

```

```

        case 6:
            System.out.println("Terima kasih telah menggunakan sistem perpustakaan!");
            // Tutup scanner sebelum keluar
            scanner.close();
            return;
        default:
            System.out.println("Pilihan tidak valid!");
    }
    System.out.println(); // Spasi antar operasi
}

// --- Metode Tampilan dan Utilitas Utama ---

private static void displayMenu() {
    System.out.println("--- SISTEM MANAJEMEN PERPUSTAKAAN ---");
    System.out.println("1. Tampilkan Semua Item");
    System.out.println("2. Tambah Item Baru");
    System.out.println("3. Pinjam Item");
    System.out.println("4. Kembalikan Item");
    System.out.println("5. Cari Item");
    System.out.println("6. Keluar");
    System.out.print("Pilih menu: ");
}

private static void initializeSampleData() {
    // Data inisialisasi menggunakan constructor dari kelas turunan
    libraryItems.add(new Book("B0001", "Pemrograman Java", 2023, "Rudi Santoso", "978-1234567890", 350));
    libraryItems.add(new Book("B0002", "Struktur Data", 2022, "Sari Dewi", "978-9876543210", 280));
    libraryItems.add(new Magazine("M001", "Info Teknologi", 2024, "Tech Media", 12, "Teknologi"));
    libraryItems.add(new DVD("D0001", "The Java Documentary", 2023, "John Programmer", 120, "Edukasi"));

    // Contoh item yang langsung dipinjam
    // Cek dulu apakah list tidak kosong
    if (!libraryItems.isEmpty()) {
        libraryItems.get(1).borrowItem();
    }
}

private static void displayAllItems() {
    System.out.println("--- DAFTAR SEMUA ITEM PERPUSTAKAAN ---");
    if (libraryItems.isEmpty()) {
        System.out.println("Perpustakaan kosong.");
        return;
    }
    for (LibraryItem item : libraryItems) {
        item.displayInfo(); // Polymorphism
        System.out.println();
    }
}

// --- Metode Operasi Perpustakaan ---

private static void addNewItem() {
    System.out.println("--- TAMBAH ITEM BARU ---");
    System.out.println("1. Buku");
    System.out.println("2. Majalah");
    System.out.println("3. DVD");
    System.out.print("Pilih jenis item: ");

    int type = 0;

```



```

if (scanner.hasNextInt()) {
    type = scanner.nextInt();
    scanner.nextLine(); // consume newline
} else {
    System.out.println("Input jenis item tidak valid. Kembali ke menu utama.");
    scanner.nextLine();
    return;
}

// Meminta data umum
System.out.print("ID Item: ");
String itemId = scanner.nextLine();
System.out.print("Judul: ");
String title = scanner.nextLine();
System.out.print("Tahun: ");
int year = 0;
if (scanner.hasNextInt()) {
    year = scanner.nextInt();
    scanner.nextLine(); // consume newline
} else {
    System.out.println("Tahun tidak valid. Kembali ke menu utama.");
    return;
}

// Meminta data spesifik berdasarkan tipe
switch (type) {
    case 1: // Book
        System.out.print("Penulis: ");
        String author = scanner.nextLine();
        System.out.print("ISBN: ");
        String isbn = scanner.nextLine();
        System.out.print("Jumlah Halaman: ");
        int pages = 0;
        if (scanner.hasNextInt()) {
            pages = scanner.nextInt();
            scanner.nextLine();
        } else {
            System.out.println("Jumlah Halaman tidak valid. Item gagal ditambahkan.");
            return;
        }

        libraryItems.add(new Book(itemId, title, year, author, isbn, pages));
        break;
    case 2: // Magazine
        System.out.print("Penerbit: ");
        String publisher = scanner.nextLine();
        System.out.print("Edisi: ");
        int issue = 0;
        if (scanner.hasNextInt()) {
            issue = scanner.nextInt();
            scanner.nextLine();
        } else {
            System.out.println("Edisi tidak valid. Item gagal ditambahkan.");
            return;
        }
        System.out.print("Kategori: ");
        String category = scanner.nextLine();

        libraryItems.add(new Magazine(itemId, title, year, publisher, issue, category));
        break;
    case 3: // DVD

```

```

        System.out.print("Sutradara: ");
        String director = scanner.nextLine();
        System.out.print("Durasi (menit): ");
        int duration = 0;
        if (scanner.hasNextInt()) {
            duration = scanner.nextInt();
            scanner.nextLine();
        } else {
            System.out.println("Durasi tidak valid. Item gagal ditambahkan.");
            return;
        }
        System.out.print("Genre: ");
        String genre = scanner.nextLine();

        libraryItems.add(new DVD(itemId, title, year, director, duration, genre));
        break;
    default:
        System.out.println("Jenis item tidak valid! Item gagal ditambahkan.");
        return;
    }

    System.out.println("Item berhasil ditambahkan!");
}

private static void borrowItem() {
    System.out.print("Masukkan ID item yang ingin dipinjam: ");
    String borrowItemId = scanner.nextLine();

    for (LibraryItem item : libraryItems) {
        if (item.getItemId().equalsIgnoreCase(borrowItemId)) {
            item.borrowItem();
            return;
        }
    }
    System.out.println("Item dengan ID " + borrowItemId + " tidak ditemukan!");
}

private static void returnItem() {
    System.out.print("Masukkan ID item yang ingin dikembalikan: ");
    String returnItemId = scanner.nextLine();

    for (LibraryItem item : libraryItems) {
        if (item.getItemId().equalsIgnoreCase(returnItemId)) {
            item.returnItem();
            return;
        }
    }
    System.out.println("Item dengan ID " + returnItemId + " tidak ditemukan!");
}

private static void searchItem() {
    System.out.print("Masukkan kata kunci pencarian (judul/penulis/sutradara): ");
    String keyword = scanner.nextLine().toLowerCase();

    System.out.println("--- HASIL PENCARIAN ---");
    boolean found = false;

    for (LibraryItem item : libraryItems) {
        // Cek di Title (berlaku untuk semua)
        if (item.getTitle().toLowerCase().contains(keyword)) {
            item.displayInfo();

```

```

        System.out.println();
        found = true;
        continue;
    }

    // Cek di atribut spesifik
    if (item instanceof Book) {
        Book book = (Book) item;
        // Asumsi field 'author' di Book adalah public atau memiliki getter (atau diakses dari class ya
        // Jika tidak, Anda perlu mengubah visibilitas di kelas Book.
        // Menggunakan asumsi visibilitas 'protected' atau 'public' untuk demonstrasi.
        // Namun, dalam kode sebelumnya, field 'author' adalah private.
        // Jika ingin mencari 'author', pastikan kelas Book memiliki getter untuk 'author'.

        // KARENA KODE BOOK SEBELUMNYA MENGGUNAKAN PRIVATE, KITA TIDAK BISA AKSES LANGSUNG.
        // Solusi: Kita tidak dapat memastikannya dari kode yang Anda berikan,
        // sehingga hanya akan mengandalkan pencarian Judul untuk menghindari error kompilasi.

        // (Untuk menghindari error, kita hanya fokus pada judul sesuai gambar awal)
    }
}

if (!found) {
    System.out.println("Tidak ada item yang sesuai dengan pencarian.");
}
}
}

```

Output

```
C:\amazon-corretto-21.0.8.9.1-windows-x64-jdk\jdk21.0.8_9\
Struktur Data berhasil dipinjam
Selamat Datang di Sistem Manajemen Perpustakaan!
--- SISTEM MANAJEMEN PERPUSTAKAAN ---
1. Tampilkan Semua Item
2. Tambah Item Baru
3. Pinjam Item
4. Kembalikan Item
5. Cari Item
6. Keluar
Pilih menu: 1
--- DAFTAR SEMUA ITEM PERPUSTAKAAN ---
--- BUKU ---
ID: B0001
Judul: Pemrograman Java
Penulis: Rudi Santoso
Tahun: 2023
ISBN: 978-1234567890
Jumlah Halaman: 350
Status: Tersedia
-----

--- BUKU ---
ID: B0002
Judul: Struktur Data
Penulis: Sari Dewi
Tahun: 2022
ISBN: 978-9876543210
Jumlah Halaman: 280
Status: Dipinjam
-----

--- MAJALAH ---
ID: M001
Judul: Info Teknologi
Penerbit: Tech Media
Tahun: 2024
Edisi: 12
Kategori: Teknologi
Status: Tersedia
-----
```

--- DVD ---

ID: D0001

Judul: The Java Documentary

Sutradara: John Programmer

Tahun: 2023

Durasi: 120 menit

Genre: Edukasi

Status: Tersedia

--- SISTEM MANAJEMEN PERPUSTAKAAN ---

1. Tampilkan Semua Item

2. Tambah Item Baru

3. Pinjam Item

4. Kembalikan Item

5. Cari Item

6. Keluar

Pilih menu: 3

Masukkan ID item yang ingin dipinjam: B0001

Pemrograman Java berhasil dipinjam

--- SISTEM MANAJEMEN PERPUSTAKAAN ---

1. Tampilkan Semua Item

2. Tambah Item Baru

3. Pinjam Item

4. Kembalikan Item

5. Cari Item

6. Keluar

Pilih menu: 4

Masukkan ID item yang ingin dikembalikan: B0001

Pemrograman Java berhasil dikembalikan

```
--- SISTEM MANAJEMEN PERPUSTAKAAN ---
1. Tampilkan Semua Item
2. Tambah Item Baru
3. Pinjam Item
4. Kembalikan Item
5. Cari Item
6. Keluar
Pilih menu: 5
Masukkan kata kunci pencarian (judul/penulis/sutradara): judul
--- HASIL PENCARIAN ---
Tidak ada item yang sesuai dengan pencarian.

--- SISTEM MANAJEMEN PERPUSTAKAAN ---
1. Tampilkan Semua Item
2. Tambah Item Baru
3. Pinjam Item
4. Kembalikan Item
5. Cari Item
6. Keluar
Pilih menu: 5
Masukkan kata kunci pencarian (judul/penulis/sutradara): The Java Documentary
--- HASIL PENCARIAN ---
--- DVD ---
ID: D0001
Judul: The Java Documentary
Sutradara: John Programmer
Tahun: 2023
Durasi: 120 menit
Genre: Edukasi
Status: Tersedia
-----

--- SISTEM MANAJEMEN PERPUSTAKAAN ---
1. Tampilkan Semua Item
2. Tambah Item Baru
3. Pinjam Item
4. Kembalikan Item
5. Cari Item
6. Keluar
Pilih menu: 6
Terima kasih telah menggunakan sistem perpustakaan!
```

Analisa dan Pembahasan

Analisa pada praktikum ini meninjau arsitektur pewarisan yang lebih kompleks, menggabungkan *Multilevel* dan *Hierarchical Inheritance* untuk membentuk struktur taksonomi yang realistis.

1. Arsitektur Pewarisan: Transitivitas dan Percabangan

Program ini membangun struktur hierarki tiga level:

`Animal → Mammal → Dog / Cat` .

- **Transitivitas Sifat (*Multilevel*):** Pada class `Dog` , meskipun ia hanya secara eksplisit melakukan `extends Mammal` , ia secara otomatis memiliki akses terhadap method `sleep()` milik `Animal` . Ini membuktikan sifat transitif dalam pewarisan: jika `Dog` adalah `Mammalia`, dan `Mammalia` adalah `Hewan`, maka secara komputasi `Dog` adalah `Hewan`. Secara struktur memori, objek `Dog` mengandung lapisan data dari `Dog` , `Mammal` , dan `Animal` .
- **Percabangan Hierarki (*Hierarchical*):** Class `Mammal` bertindak sebagai titik percabangan (*branching point*). `Dog` dan `Cat` berbagi atribut `furColor` dan perilaku `giveBirth()` yang didefinisikan di `Mammal` , namun tetap terisolasi satu sama lain dalam hal perilaku spesifik (`bark` vs `meow`). Ini menunjukkan efisiensi memori kode (*code reuse*), di mana logika umum tidak perlu diduplikasi.

2. Manajemen State Melalui Constructor Chaining 3 Level

Kompleksitas manajemen memori terlihat jelas pada saat inisialisasi objek, misalnya `new Dog("Buddy", ...)` . Terjadi eskalasi pemanggilan konstruktor berantai (*cascading constructor calls*):

1. Konstruktor `Dog` dipanggil, lalu mengeksekusi `super()` ke `Mammal` .
2. Konstruktor `Mammal` dipanggil, lalu mengeksekusi `super()` ke `Animal` .
3. Konstruktor `Animal` dieksekusi paling awal untuk menginisialisasi variabel `name` .
4. Alur kembali ke bawah (`Mammal` mengisi `furColor` , lalu `Dog` mengisi `breed`). **Analisa:** Mekanisme ini menjamin integritas data dari level teratas hingga terbawah. Setiap lapisan kelas bertanggung jawab menginisialisasi atribut yang dimilikinya sendiri, menjaga prinsip *Encapsulation* tetap utuh meskipun dalam rantai pewarisan yang panjang.

3. Polimorfisme pada Koleksi Heterogen

Pada class `InheritanceTypeTest` , terdapat implementasi array polimorfik: `Animal[] animals = {new Dog(...), new Cat(...)};` . Di sini terjadi fenomena **Upcasting** implisit, di mana objek `Dog` dan `Cat` dipandang sebagai tipe umum `Animal` . Analisa mendalam pada saat *runtime*:

- **Compile-Time:** Compiler hanya memverifikasi bahwa method `eat()` ada di class `Animal` .
- **Run-Time (Dynamic Dispatch):** Ketika loop berjalan, JVM menelusuri *Virtual Method Table* (vtable) dari objek yang sedang diakses. Untuk indeks 0, vtable menunjuk ke implementasi `Dog.eat()` , dan untuk indeks 1 ke `Cat.eat()` .
- **Hasil:** Perilaku program menjadi adaptif. Meskipun perintah pemanggilannya sama (`animal.eat()`), output yang dihasilkan berbeda ("eating dog food" vs "eating cat food"). Ini membuktikan fleksibilitas sistem di mana satu antarmuka (`Animal`) dapat menangani berbagai bentuk implementasi objek secara dinamis.

Kesimpulan

Berdasarkan serangkaian percobaan yang telah dilakukan dari Praktikum 1 hingga Praktikum 4, dapat ditarik beberapa kesimpulan utama mengenai implementasi Pewarisan dan Polimorfisme dalam Java:

1. Efisiensi dan Reusabilitas Kode: Konsep Inheritance terbukti efektif dalam mengurangi redundansi kode. Atribut dan method yang bersifat umum cukup didefinisikan satu kali pada superclass (seperti `Person`, `Vehicle`, atau

- LibraryItem), dan dapat digunakan kembali oleh berbagai subclass, sehingga struktur program menjadi lebih ringkas dan terorganisir.
2. Manajemen State dengan Constructor Chaining: Penggunaan kata kunci `super()` sangat krusial dalam hierarki pewarisan bertingkat (multilevel inheritance). Hal ini menjamin bahwa inisialisasi variabel pada superclass dilakukan terlebih dahulu sebelum subclass menginisialisasi atribut spesifiknya, menjaga integritas data objek secara keseluruhan.
 3. Fleksibilitas melalui Polimorfisme dan Overriding: Method Overriding memberikan fleksibilitas bagi subclass untuk memodifikasi perilaku yang diwarisi sesuai kebutuhan spesifiknya (spesialisasi). Dikombinasikan dengan Polymorphism, sistem memungkinkan pemrosesan objek secara generik (misalnya menggunakan ArrayList), namun eksekusi program tetap mengikuti logika spesifik dari tipe objek aslinya saat runtime (dynamic binding).
 4. Hierarki Kelas sebagai Solusi Kompleksitas: Penerapan Multilevel dan Hierarchical Inheritance memungkinkan pemodelan sistem dunia nyata yang kompleks (seperti taksonomi hewan atau sistem perpustakaan) ke dalam struktur kode yang logis. Pemisahan antara atribut umum dan spesifik melalui pewarisan memudahkan pengembangan fitur baru tanpa mengganggu kode yang sudah ada.

Referensi

1. Oracle. (2023). The Java™ Tutorials: Inheritance. Oracle Documentation. Diakses dari <https://docs.oracle.com/javase/tutorial/java/land/subclasses.html>
2. Schildt, H. (2018). Java: The Complete Reference (11th ed.). McGraw-Hill Education.
3. Eckel, B. (2006). Thinking in Java (4th ed.). Prentice Hall.
4. Modul Praktikum Pemrograman Berorientasi Objek. (2025). Modul 6: Inheritance dan Polymorphism. Jurusan Teknologi Informasi dan Komputer, Politeknik Negeri Lhokseumawe.