PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

INTERFACE DAN MULTIPLE INHERITANCE

Anggota Kelompok :

Rizky Satria Gunawan

(241511089)

2C-D3

PROGRAM STUDI D3 TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2025

Link github : RizkySatria123/Mata-Kuliah-PBO: Untuk Mengumpulan tugas PBO

Task 3: Multiple Inheritance melalui Interface

Overview

Task 3 mendemonstrasikan bagaimana Java mengatasi keterbatasan single inheritance dengan menggunakan multiple interfaces. Implementasi ini menunjukkan:

- Multiple interface implementation
- Interface-based polymorphism
- Kombinasi inheritance dan interface implementation
- Realistic employee management system

1.1 Interface Definitions

Payable Interface

```java
package task3;

/**
 * Payable interface: any class that can compute payment.
 */
public interface Payable {
    double computePay();
}
```

WorkLog Interface

```java
package task3;

/**
 * WorkLog interface for tracking worked hours.
 */
public interface WorkLog {
    void addHours(double hours);
    double getTotalHours();
}
```

Trainable Interface

```java
package task3;
```

```java
/**
 * Trainable interface for entities that can attend training.
 */
public interface Trainable {
    void attendTraining(String topic);
    int getTrainingCount();
}
```

Key Features:

- **Single Responsibility:** Setiap interface fokus pada satu aspek functionality
- **Contract Definition:** Interface mendefinisikan "what" bukan "how"
- **Multiple Implementation:** Classes dapat implement multiple interfaces

1.2 PersonBase - Simple Superclass

```java
package task3;

/**
 * PersonBase as a simple superclass reused by Employee and others.
 */
public class PersonBase {
    private final String name;

    public PersonBase(String name) {
        this.name = name == null ? "Unknown" : name;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "PersonBase[name=" + name + "]";
    }
}
```

Key Features:

- **Immutable Name:** Final field dengan null safety

- **Base Functionality:** Common behavior untuk semua person-based classes
- **Simple Design:** Minimal interface sebagai foundation

1.3 Employee - Multiple Interface Implementation

```java
package task3;

import java.util.ArrayList;
import java.util.List;

/**
 * Employee demonstrates multiple inheritance via implementing multiple
interfaces.
 */
public class Employee extends PersonBase implements Payable, WorkLog, Trainable {
    private final double hourlyRate;
    private double totalHours;
    private final List<String> trainings = new ArrayList<>();

    public Employee(String name, double hourlyRate) {
        super(name);
        if (hourlyRate < 0) {
            throw new IllegalArgumentException("Hourly rate must be non-
negative");
        }
        this.hourlyRate = hourlyRate;
    }

    @Override
    public double computePay() {
        return hourlyRate * totalHours;
    }

    @Override
    public void addHours(double hours) {
        if (hours < 0) {
            throw new IllegalArgumentException("Hours must be non-negative");
        }
        totalHours += hours;
    }

    @Override
    public double getTotalHours() {
        return totalHours;
    }

    @Override
    public void attendTraining(String topic) {
        trainings.add(topic == null ? "(unspecified)" : topic);
    }

    @Override
    public int getTrainingCount() {
        return trainings.size();
    }

    public double getHourlyRate() { return hourlyRate; }

    public List<String> getTrainings() { return List.copyOf(trainings); }

    @Override
    public String toString() {
        return
String.format("Employee[%s,rate=%.2f,hours=%.2f,pay=%.2f,trainings=%d]",
super.toString(), hourlyRate, totalHours, computePay(), trainings.size());
    }
}
```

Key Features:

- **Multiple Interface Implementation:** Implements Payable, WorkLog, dan Trainable
- **State Management:** Tracks hours, rate, dan training history
- **Defensive Copying:** getTrainings() returns immutable view
- **Comprehensive Validation:** Input validation untuk semua parameters

1.4 Manager - Inheritance dengan Method Override

```
package task3;

/**
 * Manager extends Employee adding bonus and overriding computePay while using
super.computePay().
 */
public class Manager extends Employee {
    private double bonus; // one-time bonus added to pay
```

```java
    public Manager(String name, double hourlyRate, double bonus) {
        super(name, hourlyRate);
        if (bonus < 0) {
            throw new IllegalArgumentException("Bonus must be non-negative");
        }
        this.bonus = bonus;
    }

    public double getBonus() { return bonus; }
    public void setBonus(double bonus) { if (bonus < 0) throw new
IllegalArgumentException("Bonus must be non-negative"); this.bonus = bonus; }

    @Override
    public double computePay() {
        return super.computePay() + bonus; // augment base logic
    }

    @Override
    public String toString() {
        return "Manager{" + super.toString() + ",bonus=" + bonus + '}';
    }
}
```

Key Features:

- **Extends Employee:** Inherits semua interface implementations
- **Method Override:** computePay() augments parent behavior dengan bonus
- **Bonus Management:** Additional functionality specific untuk managers
- **super() Usage:** Proper delegation ke parent implementation

1.5 Trainer - Specialized Employee

```java
package task3;

/**
 * Trainer is a specialized Employee focusing on training others.
 */
public class Trainer extends Employee {
    private int sessionsLed;

    public Trainer(String name, double hourlyRate) {
        super(name, hourlyRate);
    }

    public void leadSession(String topic) {
        sessionsLed++;
        // Reuse attendTraining to log own session as attended knowledge
        attendTraining("Led: " + topic);
    }

    public int getSessionsLed() { return sessionsLed; }

    @Override
    public String toString() {
        return "Trainer{" + super.toString() + ",sessionsLed=" + sessionsLed +
'}';
    }
}
```

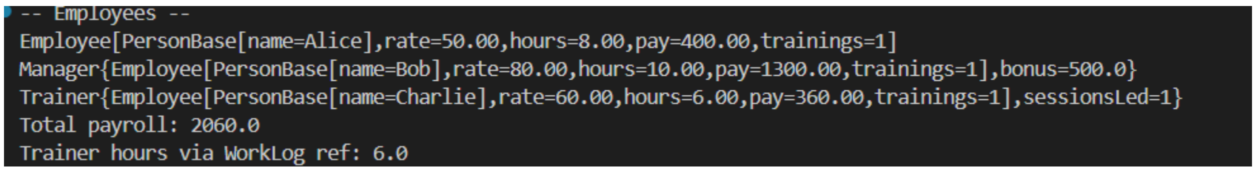
Key Features:

- **Specialized Behavior:** leadSession() specific untuk training role
- **Method Reuse:** Clever reuse of inherited attendTraining() method
- **Domain Logic:** Realistic business logic untuk trainer functionality

1.6 Demo dan Polymorphism Testing

```
package task3;

/**
 * Demo for multiple inheritance via interfaces and class hierarchy.
 */
public class Task3Demo {
    public static void main(String[] args) {
        Employee e = new Employee("Alice", 50);
        e.addHours(8);
        e.attendTraining("Safety");

        Manager m = new Manager("Bob", 80, 500);
        m.addHours(10);
        m.attendTraining("Leadership");

        Trainer t = new Trainer("Charlie", 60);
        t.addHours(6);
        t.leadSession("Java Inheritance");
        System.out.println("-- Employees --");
        System.out.println(e);
        System.out.println(m);
        System.out.println(t);

        // Polymorphism through interface references
        Payable[] payroll = { e, m, t };
        double total = 0;
        for (Payable p : payroll) {
            total += p.computePay();
        }
        System.out.println("Total payroll: " + total);

        // WorkLog reference
        WorkLog wl = t; // Trainer implements WorkLog via Employee
        System.out.println("Trainer hours via WorkLog ref: " +
wl.getTotalHours());
    }
}
```

Expected Output:



```
-- Employees --
Employee[PersonBase[name=Alice],rate=50.00,hours=8.00,pay=400.00,trainings=1]
Manager{Employee[PersonBase[name=Bob],rate=80.00,hours=10.00,pay=1300.00,trainings=1],bonus=500.0}
Trainer{Employee[PersonBase[name=Charlie],rate=60.00,hours=6.00,pay=360.00,trainings=1],sessionsLed=1}
Total payroll: 2060.0
Trainer hours via WorkLog ref: 6.0
```

1.7 Solusi untuk Multiple Inheritance Problem

Problem dari Exercise: Java tidak mendukung multiple class inheritance seperti class Manager extends Employee extends Sortable.

Solusi yang Diimplementasikan:

1. **Interface-based Multiple Inheritance:**
2. class Employee extends PersonBase implements Payable, WorkLog, Trainable
3. **Composition over Inheritance:**

   - Interfaces define contracts
   - Classes implement multiple contracts
   - Behavior composition melalui interface implementation
4. **Mixin Pattern Alternative:**
5. // Trainer dapat act sebagai Employee, Payable, WorkLog, dan Trainable
6. Employee emp = new Trainer("John", 70);
7. Payable payable = emp;
8. WorkLog workLog = emp;
9. Trainable trainable = emp;

1.8 Advanced Polymorphism Patterns

Interface Segregation

```
// Different views of same object
Employee emp = new Manager("Sarah", 90, 1000);

// Payroll system hanya perlu Payable interface
Payable payableView = emp;
double pay = payableView.computePay();

// HR system hanya perlu Trainable interface
Trainable trainableView = emp;
trainableView.attendTraining("Management Skills");

// Time tracking system hanya perlu WorkLog interface
```

```
WorkLog workLogView = emp;
workLogView.addHours(8.5);
```

Dynamic Type Resolution

```
public static void processEmployee(Employee emp) {
    System.out.println("Processing: " + emp);

    if (emp instanceof Manager mgr) {
        System.out.println("Manager bonus: " + mgr.getBonus());
    } else if (emp instanceof Trainer trainer) {
        System.out.println("Sessions led: " + trainer.getSessionsLed());
    }

    // Interface methods work regardless of concrete type
    System.out.println("Pay: " + emp.computePay());
    System.out.println("Hours: " + emp.getTotalHours());
}
```

Soal 5 Interface

Praktik ini bertujuan untuk mendemonstrasikan konsep inti Pemrograman Berorientasi Objek (PBO) dengan membuat sebuah hierarki kelas yang merepresentasikan berbagai jenis barang (`Goods`) beserta turunannya. Selain pewarisan, digunakan juga interface `Taxable` untuk menambahkan perilaku spesifik (perhitungan pajak) pada kelas tertentu.

Tujuan

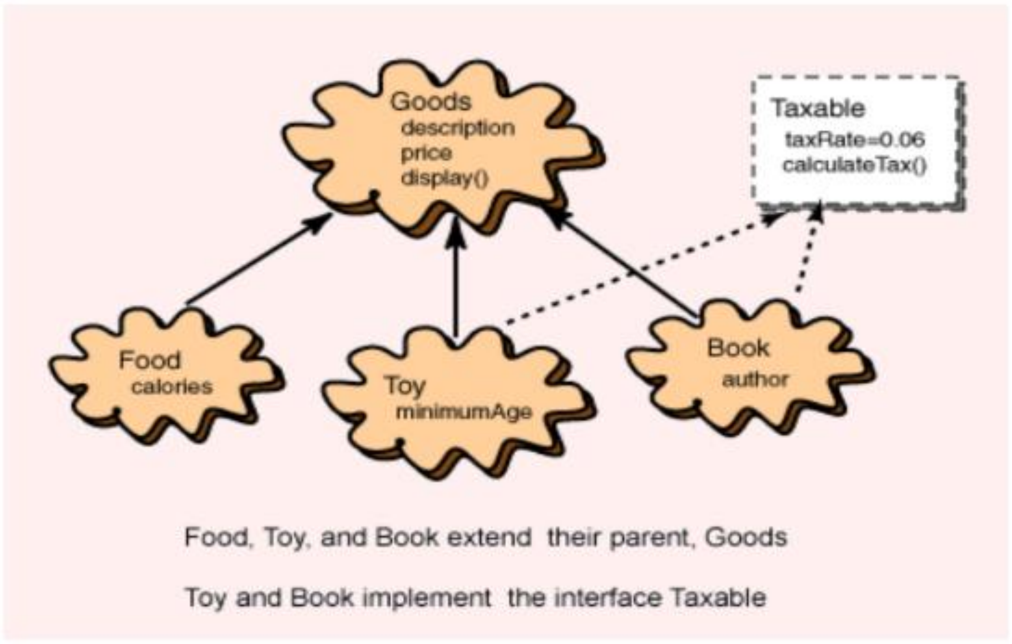
- Mengimplementasikan inheritance antara superclass dan subclass.
- Menggunakan interface sebagai kontrak perilaku.
- Menunjukkan polymorphism melalui overriding method dan implementasi interface.
- Menerapkan enkapsulasi melalui penggunaan atribut `private` dan akses melalui getter/setter.
- Menyusun eksekusi program yang memanfaatkan struktur hierarki secara terorganisir.

Desain dan Struktur Kode

Struktur paket: `task4`

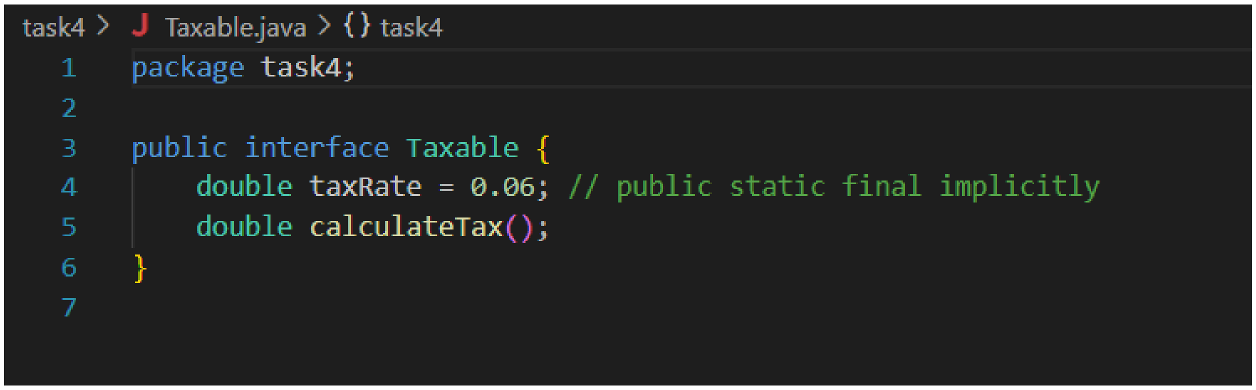
Kelas & Interface:

- `Goods` (superclass)
- `Food`, `Toy`, `Book` (subclass `extends Goods`)
- `Taxable` (interface: diimplementasikan oleh `Toy` dan `Book`)
- `Main` (driver / kelas uji)



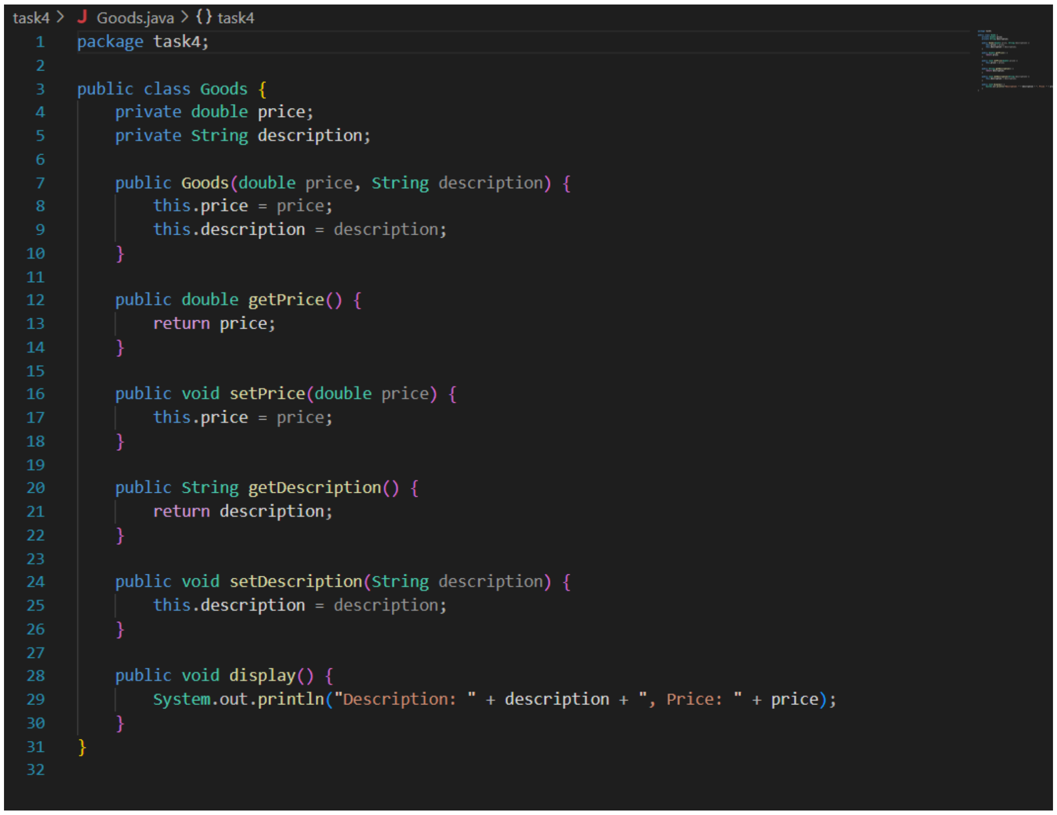
Implementasi Kode (Potongan Inti)

1 Interface `Taxable`


task4 > Taxable.java > {} task4

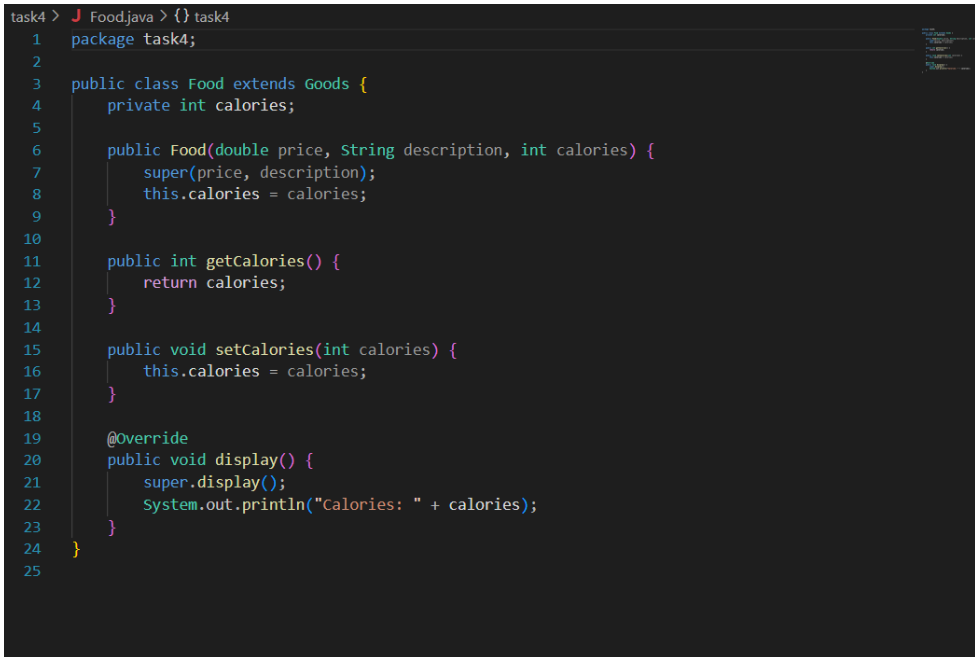
```java
package task4;

public interface Taxable {
    double taxRate = 0.06; // public static final implicitly
    double calculateTax();
}

```

2. Superclass `Goods`


task4 > Goods.java > {} task4

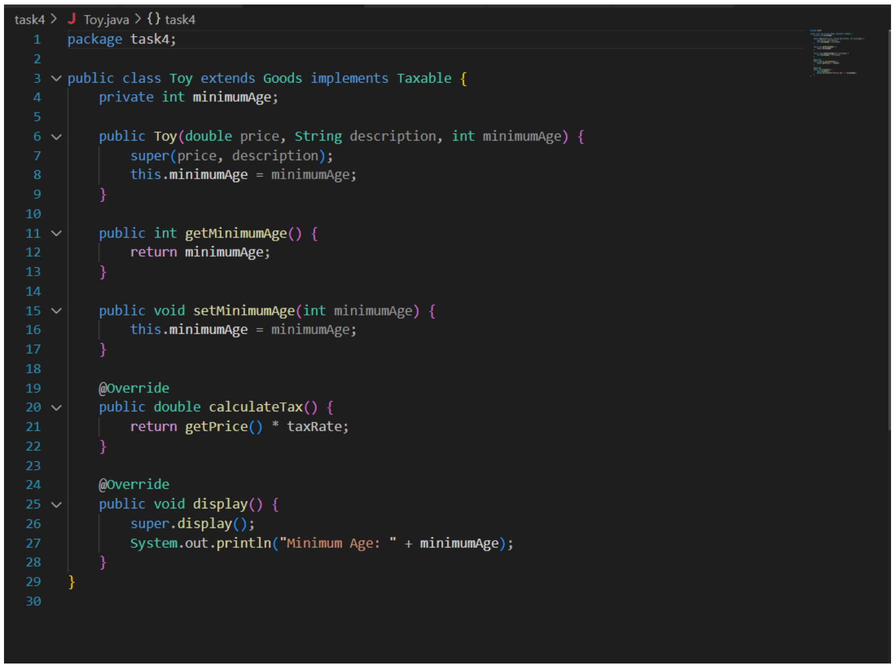
```java
package task4;

public class Goods {
    private double price;
    private String description;

    public Goods(double price, String description) {
        this.price = price;
        this.description = description;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void display() {
        System.out.println("Description: " + description + ", Price: " + price);
    }
}
```

3. Subclass `Food`


task4 > Food.java > {} task4

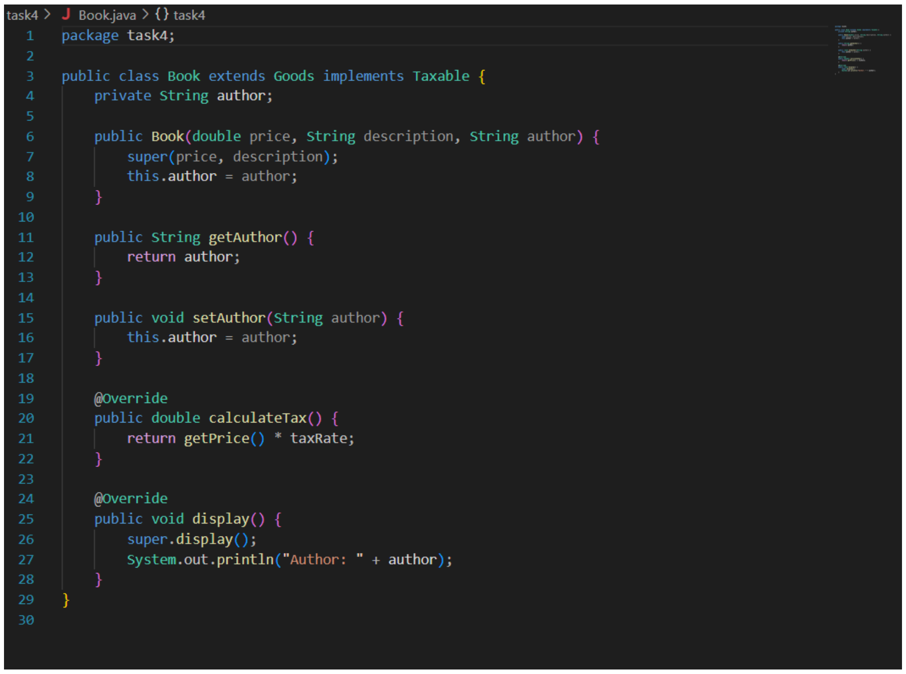
```java
package task4;

public class Food extends Goods {
    private int calories;

    public Food(double price, String description, int calories) {
        super(price, description);
        this.calories = calories;
    }

    public int getCalories() {
        return calories;
    }

    public void setCalories(int calories) {
        this.calories = calories;
    }

    @Override
    public void display() {
        super.display();
        System.out.println("Calories: " + calories);
    }
}
```

4. Subclass `Toy` (implements `Taxable`)


task4 > Toy.java > {} task4

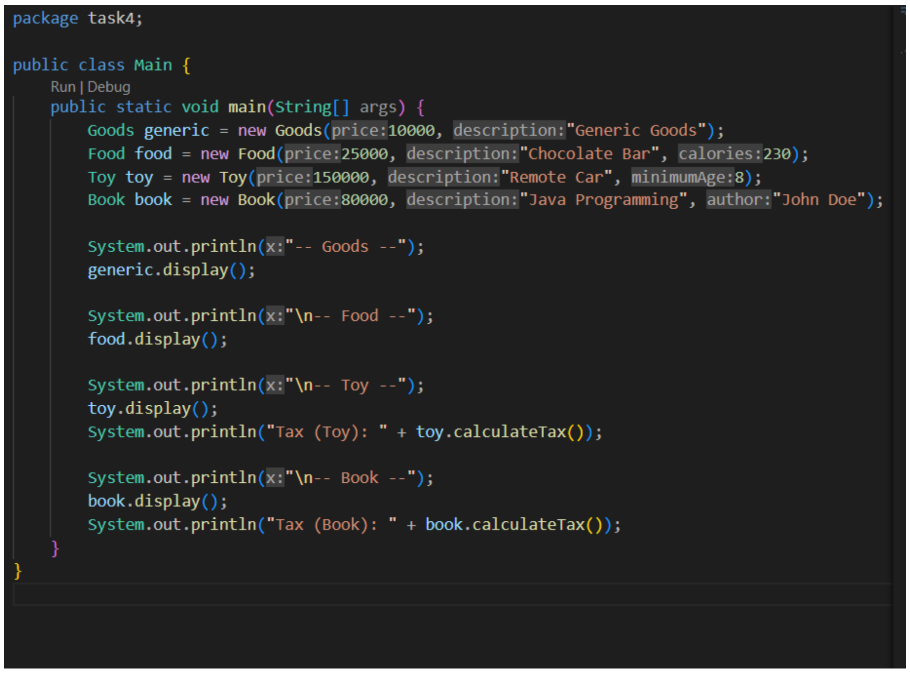
```java
package task4;

public class Toy extends Goods implements Taxable {
    private int minimumAge;

    public Toy(double price, String description, int minimumAge) {
        super(price, description);
        this.minimumAge = minimumAge;
    }

    public int getMinimumAge() {
        return minimumAge;
    }

    public void setMinimumAge(int minimumAge) {
        this.minimumAge = minimumAge;
    }

    @Override
    public double calculateTax() {
        return getPrice() * taxRate;
    }

    @Override
    public void display() {
        super.display();
        System.out.println("Minimum Age: " + minimumAge);
    }
}
```

5. Subclass `Book` (implements `Taxable`)


task4 > Book.java > {} task4

```java
package task4;

public class Book extends Goods implements Taxable {
    private String author;

    public Book(double price, String description, String author) {
        super(price, description);
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    @Override
    public double calculateTax() {
        return getPrice() * taxRate;
    }

    @Override
    public void display() {
        super.display();
        System.out.println("Author: " + author);
    }
}
```

6. Kelas `Main`



```java
package task4;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Goods generic = new Goods(price:10000, description:"Generic Goods");
        Food food = new Food(price:25000, description:"Chocolate Bar", calories:230);
        Toy toy = new Toy(price:150000, description:"Remote Car", minimumAge:8);
        Book book = new Book(price:80000, description:"Java Programming", author:"John Doe");

        System.out.println(x:"-- Goods --");
        generic.display();

        System.out.println(x:"\n-- Food --");
        food.display();

        System.out.println(x:"\n-- Toy --");
        toy.display();
        System.out.println("Tax (Toy): " + toy.calculateTax());

        System.out.println(x:"\n-- Book --");
        book.display();
        System.out.println("Tax (Book): " + book.calculateTax());
    }
}
```

Analisis Konsep PBO

1. Inheritance

`Food`, `Toy`, dan `Book` mewarisi atribut dan metode umum (`price`, `description`, `display()`) dari `Goods`. Konstruktor subclass memanggil `super(...)` sehingga pewarisan state berjalan benar.

2. Interface & Kontrak Perilaku

`Taxable` memaksakan adanya metode `calculateTax()`. Hanya kelas yang logis dikenai pajak (`Toy` dan `Book`) yang mengimplementasikan interface ini — memisahkan aspek pajak dari struktur inti `Goods`.

3. Polymorphism

- Overriding `display()` di tiap subclass menambah perilaku khusus sambil mempertahankan perilaku dasar (`super.display()`).

- Implementasi `calculateTax()` di `Toy` dan `Book` meski identik secara formula, tetap menunjukkan polimorfisme karena dipanggil melalui tipe interface (`Taxable`). Potensi perluasan: rumus pajak berbeda per kategori tanpa ubah kode klien.

4. Encapsulation

Semua atribut bersifat `private`. Akses dimediasi lewat getter/setter sehingga validasi atau perubahan representasi internal dapat ditambahkan tanpa memengaruhi kode pemanggil.

Eksekusi Program

1. Pembuatan empat objek dengan constructor masing-masing.

2. Pemanggilan `display()` menampilkan informasi berbeda sesuai tipe runtime (overriding).

3. Untuk objek yang `instanceof Taxable`, metode `calculateTax()` dipanggil dan hasilnya dicetak.

4. Alur linear, mudah ditelusuri, mendemonstrasikan penggunaan hierarki.

Hasil Uji Eksekusi (Output)

```
-- Goods --

Description: Generic Goods, Price: 10000.0



-- Food --

Description: Chocolate Bar, Price: 25000.0

Calories: 230
```

-- Toy --

Description: Remote Car, Price: 150000.0

Minimum Age: 8

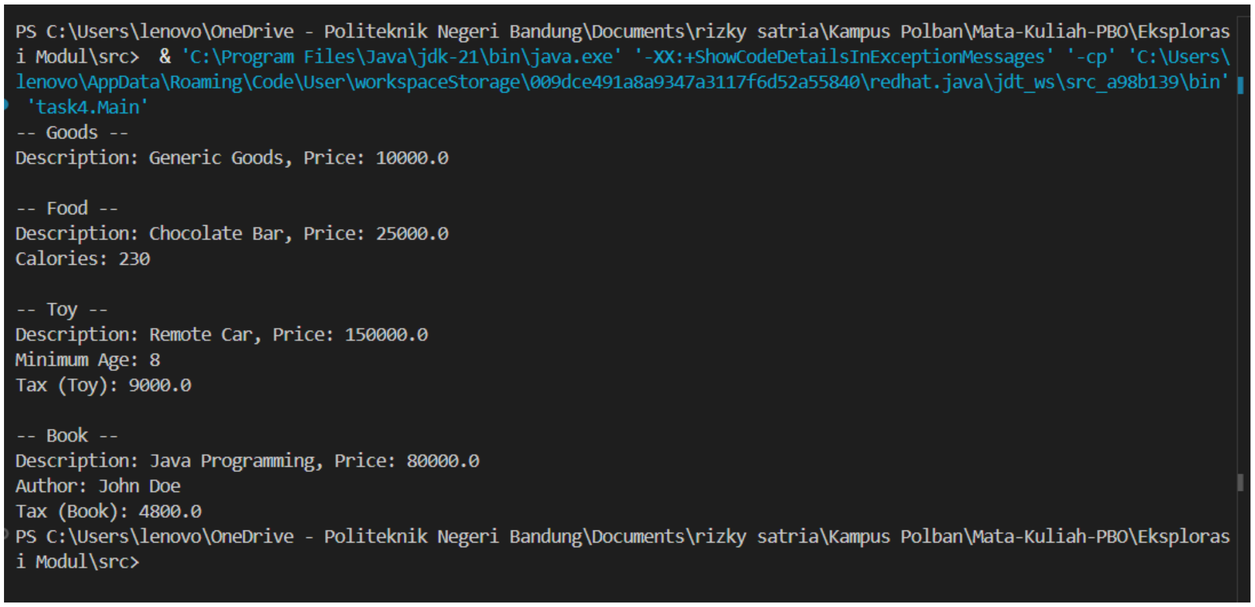
Tax (Toy): 9000.0

-- Book --

Description: Java Programming, Price: 80000.0

Author: John Doe

Tax (Book): 4800.0

```



```
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\Eksploras
i Modul\src> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\
lenovo\AppData\Roaming\Code\User\workspaceStorage\009dce491a8a9347a3117f6d52a55840\redhat.java\jdt_ws\src_a98b139\bin'
 'task4.Main'
-- Goods --
Description: Generic Goods, Price: 10000.0

-- Food --
Description: Chocolate Bar, Price: 25000.0
Calories: 230

-- Toy --
Description: Remote Car, Price: 150000.0
Minimum Age: 8
Tax (Toy): 9000.0

-- Book --
Description: Java Programming, Price: 80000.0
Author: John Doe
Tax (Book): 4800.0
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\Eksploras
i Modul\src>
```

## Kesimpulan

Praktik ini berhasil mendemonstrasikan penerapan empat pilar PBO: inheritance, interface (sebagai bentuk abstraksi tambahan), polymorphism melalui overriding dan kontrak interface, serta encapsulation lewat akses terkontrol. Struktur yang dibangun fleksibel untuk pengembangan
```

fitur lanjutan seperti strategi pajak berbeda, penyimpanan dalam koleksi, atau integrasi dengan antarmuka pengguna.