

**LAPORAN**  
**WEEK 5 PEMROGRAMAN BERBASIS OBJEK**

Dibuat untuk memenuhi salah satu tugas mata kuliah Pemrograman Berbasis Objek yang diampu  
oleh Bapak Ardhian Ekawijana, M.T.

Oleh:

Nama	: Raihana Aisha Az-Zahra
NIM	: 241511056
Kelas	: 2B
Program Studi	: D3 Teknik Informatika
Jurusan	: Teknik Komputer dan Informatika



**POLITEKNIK NEGERI BANDUNG**  
**KOTA BANDUNG**  
**2025**

## Daftar Isi



<b>BAB I HASIL Pengerjaan</b> .....	<b>3</b>
<b>1. Hasil Pengerjaan Task 3.1</b> .....	<b>3</b>
a. Case 1 .....	3
b. Hasil Case 1 .....	4
c. Case 2 .....	4
d. Hasil Case 2 .....	5
<b>2. Hasil Pengerjaan Soal 5 Interface</b> .....	<b>5</b>
a. Interface Taxable .....	5
b. Goods (Parent Class) .....	6
c. Food (Subclass) .....	7
d. Toy (Subclass) .....	7
e. Book (Subclass) .....	8
f. GoodsTest (Main Class) .....	8
g. Hasil Program Goods .....	9
<b>BAB II LESSON LEARNED</b> .....	<b>10</b>

# BAB I

## HASIL Pengerjaan

### 1. Hasil Pengerjaan Task 3.1

#### a. Case 1

Program	Penjelasan
 <pre> 1  abstract class Sortable{ 2      public abstract int compare(Sortable b); 3 4      public static void shell_sort(Sortable[] a) { 5          int n = a.length; 6          for (int gap = n / 2; gap &gt; 0; gap /= 2) { 7              for (int i = gap; i &lt; n; i++) { 8                  Sortable temp = a[i]; 9                  int j = i; 10                 while (j &gt;= gap &amp;&amp; a[j - gap].compare(temp) &gt; 0) { 11                     a[j] = a[j - gap]; 12                     j -= gap; 13                 } 14                 a[j] = temp; 15             } 16         } 17     } 18 } </pre>	<p>Sortable adalah abstract class yang mendefinisikan kontrak method compare(Sortable b) untuk membandingkan objek. Method shell_sort statis bisa mengurutkan array objek apapun yang mewarisi Sortable dengan memanfaatkan compare. Hal ini membuat sorting bersifat umum, tanpa tergantung tipe objek spesifik.</p>
 <pre> 1  class Employee extends Sortable{ 2 3      public Employee(String n, double s, int day, int month, int year){ 4          name = n; 5          salary = s; 6          hireday = day; 7          hiremonth = month; 8          hireyear = year; 9      } 10 11     public void print(){ 12         System.out.println(name + " " + salary + " " + hireYear()); 13     } 14 15     public void raiseSalary(double byPercent){ 16         salary *= 1 + byPercent / 100; 17     } 18 19     public int hireYear(){ 20         return hireyear; 21     } 22 23     public int compare(Sortable b){ 24         Employee eb = (Employee) b; 25         if(salary&gt;eb.salary) return -1; 26         if(salary&lt;eb.salary) return 1; 27         return 0; 28     } 29 30     private String name; 31     private double salary; 32     private int hireday; 33     private int hiremonth; 34     private int hireyear; 35 36 } </pre>	<p>Employee mewarisi Sortable dan mengimplementasikan method compare berdasarkan salary. Objek Employee sekarang bisa dibandingkan satu sama lain. Method raiseSalary digunakan untuk menaikkan gaji, sedangkan print menampilkan info lengkap. Ini memungkinkan array Employee[] diurutkan menggunakan shell_sort.</p>

<pre> 1 public class EmployeeTest{ 2     public static void main(String[] args) { 3         Employee[] staff = new Employee[3]; 4         staff[0] = new Employee("Raihana Aisha", 2000000, 1, 10, 1989); 5         staff[1] = new Employee("Dimas Chairil", 2500000, 1, 12, 1991); 6         staff[2] = new Employee("Budi Susanto", 3000000, 1, 11, 1993); 7 8         System.out.println("Compare staff[0] with staff[1] "+ staff[0].compare(staff[1])); 9         Sortable.shell_sort(staff); 10 11 12         int i; 13         for(i = 0; i &lt; 3; i++) staff[i].raiseSalary(5); 14         for(i = 0; i &lt; 3; i++) staff[i].print(); 15     } 16 } </pre>	<p>Pada kode ini, array Employee[] dibuat dan memanggil compare untuk melihat perbandingan gaji antar objek. Setelah itu, shell_sort mengurutkan array berdasarkan gaji, dan setiap gaji dinaikkan 5%. Blok ini menunjukkan penggunaan Sortable secara nyata pada objek Employee.</p>
---	---

## b. Hasil Case 1

```

PS D:\Semester 3\PBO Praktek\Task 3> cd
st }
Compare staff[0] with staff[1] -1
Raihana Aisha 2100000.0 1989
Dimas Chairil 2625000.0 1991
Budi Susanto 3150000.0 1993

```

## c. Case 2

Untuk Case 2, Java tidak mendukung multiple inheritance untuk class, yang artinya sebuah class tidak bisa mewarisi lebih dari satu class secara langsung. Misalnya, *class Manager extends Employee extends Sortable* tidak bisa dilakukan, karena Java hanya mengizinkan satu class parent pada keyword extends. Hal ini dilakukan untuk menghindari ambiguitas dan konflik yang bisa muncul jika dua parent class memiliki method atau field dengan nama yang sama, sehingga compiler tidak tahu mana yang harus diwarisi.

Solusinya adalah menggunakan single inheritance yang tetap memanfaatkan hierarki yang ada. Karena Employee sudah extends Sortable, otomatis semua subclass dari Employee, termasuk Manager, sudah menjadi turunan dari Sortable juga.

Program	Penjelasan
	<p>Manager mewarisi semua atribut dan method dari Employee, termasuk compare. Ini memungkinkan Manager langsung bisa dibandingkan dengan Employee. Method raiseSalary di-override untuk menambahkan bonus</p>

<pre> 1 import java.util.Calendar; 2 import java.util.GregorianCalendar; 3 4 class Manager extends Employee{ 5 6     public Manager(String n, double s, int d, int m, int y){ 7         super(n,s,d,m,y); 8         secretaryName = ""; 9     } 10 11     @Override 12     public void raiseSalary(double byPercent){ 13         // tambah 1/2% bonus untuk layanan setiap tahun 14         GregorianCalendar todaysDate = new GregorianCalendar(); 15         int currentYear = todaysDate.get(Calendar.YEAR); 16         double bonus = 0.5 * (currentYear - hireYear()); 17         super.raiseSalary(byPercent+bonus); 18     } 19 20     public String getSecretaryName(){ 21         return secretaryName; 22     } 23     private String secretaryName; 24 } </pre>	<p>0.5% per tahun pengalaman. secretaryName ditambahkan sebagai atribut khusus Manager.</p>
<pre> 1 public class ManagerTest{ 2     public static void main(String[] args) { 3         Employee[] staff = new Employee[3]; 4         staff[0] = new Employee("Raihana Aisha", 2000000, 1, 10, 1989); 5         staff[1] = new Employee("Dimas Chairil", 2500000, 1, 12, 1991); 6         staff[2] = new Manager("Budi Susanto", 3000000, 1, 11, 1993); 7 8         System.out.println("Compare staff[0] vs staff[1]: " + staff[0].compare(staff[1])); 9         System.out.println("Compare staff[1] vs staff[2]: " + staff[1].compare(staff[2])); 10        System.out.println("Compare staff[2] vs staff[0]: " + staff[2].compare(staff[0])); 11 12        int i; 13        for(i = 0; i &lt; 3; i++) staff[i].raiseSalary(5); 14        for(i = 0; i &lt; 3; i++) staff[i].print(); 15    } 16 } </pre>	<p>Di sini array Employee[] berisi Employee biasa dan satu Manager. Panggilan compare menunjukkan bahwa Manager tetap bisa dibandingkan berdasarkan gaji tanpa harus multiple inheritance. Method raiseSalary juga berlaku, termasuk bonus khusus Manager, dan print menampilkan semua data.</p>

#### d. Hasil Case 2

```

PS D:\Semester 3\PBO Praktek\Task 3> cd
}
Compare staff[0] vs staff[1]: -1
Compare staff[1] vs staff[2]: -1
Compare staff[2] vs staff[0]: 1
Raihana Aisha 2100000.0 1989
Dimas Chairil 2625000.0 1991
Budi Susanto 3630000.0 1993

```

## 2. Hasil Pengerjaan Soal 5 Interface

### a. Interface Taxable

```

interface Taxable{
    double taxRate = 0.06;
    double calculateTax();
}

```

Bagian ini adalah sebuah interface bernama Taxable yang berfungsi sebagai kontrak. Interface ini mendefinisikan sebuah konstanta taxRate dengan nilai 6% dan sebuah method abstrak calculateTax() yang wajib diimplementasikan oleh setiap class yang menggunakan interface ini. Artinya, class yang implements Taxable harus menyediakan cara menghitung pajaknya sendiri, meskipun rumus dasarnya sama yaitu harga \* taxRate. Dengan cara ini, kita bisa memastikan hanya class tertentu yang bisa dikenai pajak, yaitu Toy dan Book.

#### b. Goods (Parent Class)

```
abstract class Goods{
    private String description;
    private double price;

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

    public String getDesc(){
        return description;
    }

    public double getPrice(){
        return price;
    }

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

Goods adalah class induk yang menjadi dasar bagi semua barang. Class ini memiliki dua atribut utama yaitu description untuk deskripsi barang, dan price untuk menyimpan harga. Keduanya diset melalui constructor. Tersedia getter (getDescription dan getPrice) untuk mengambil data, serta method display() yang menampilkan informasi barang ke layar. Semua subclass seperti Food, Toy, dan Book akan mewarisi atribut dan perilaku ini sehingga tidak perlu menuliskannya ulang.

### c. Food (Subclass)

```
class Food extends Goods{
    private int calories;

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

    public int getCalories(){
        return calories;
    }

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

Food adalah subclass dari Goods yang merepresentasikan makanan. Selain atribut description dan price yang diwarisi dari Goods, Food menambahkan atribut khusus yaitu calories untuk menyimpan jumlah kalori makanan. Constructor Food menggunakan super(description, price) untuk memanggil constructor Goods agar atribut dasar bisa diisi, lalu menambahkan pengisian calories. Dengan ini, setiap objek Food tidak hanya punya harga dan deskripsi, tetapi juga nilai kalori.

### d. Toy (Subclass)

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

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

    public int getMinimumAge(){
        return minimumAge;
    }

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

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

Toy adalah subclass dari Goods yang sekaligus mengimplementasikan interface Taxable. Hal ini berarti mainan (Toy) tidak hanya memiliki atribut umum barang seperti deskripsi dan harga, tetapi juga bisa dihitung pajaknya. Atribut khusus yang dimiliki Toy adalah minimumAge untuk menentukan batas usia minimal anak yang boleh

memainkannya. Method `calculateTax()` diimplementasikan sesuai kontrak dari `Taxable`, yaitu menghitung pajak sebesar `price * taxRate`.

#### e. Book (Subclass)

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

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

    public String getAuthor(){
        return author;
    }

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

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

Book adalah subclass lain dari `Goods` yang juga implements `Taxable`, sehingga buku bisa dikenakan pajak. Atribut tambahan yang dimiliki Book adalah `author` untuk menyimpan nama penulis buku. Sama seperti pada `Toy`, class ini wajib mengimplementasikan `calculateTax()` untuk menghitung pajak berdasarkan harga buku. Dengan demikian, Book dapat menampilkan informasi dasar seperti deskripsi dan harga, sekaligus informasi tambahan yaitu penulis dan nilai pajak yang dikenakan.

#### f. GoodsTest (Main Class)

```
public class GoodsTest{
    Run main | Debug main
    public static void main(String[] args) {
        Goods apple = new Food("apple", 2.0, 95);
        Goods lego = new Toy("Lego", 50.0, 6);
        Goods book = new Book("Java Programming", 30.0, "John");

        apple.display();
        System.out.println();
        lego.display();
        System.out.println();
        book.display();
    }
}
```

`GoodsTest` adalah class utama untuk menguji semua class yang sudah dibuat. Di sini dibuat tiga objek: `Food`, `Toy`, dan `Book`. Masing-masing objek menampilkan informasi



dasar dari parent class (display()), kemudian menampilkan atribut khususnya (calories untuk makanan, minimumAge untuk mainan, dan author untuk buku). Untuk Toy dan Book, method calculateTax() dipanggil untuk menghitung pajak sesuai tarif 6%. Dengan begitu, output yang dihasilkan akan menunjukkan kombinasi antara data induk (Goods), data tambahan dari subclass, dan hasil perhitungan pajak untuk barang yang dikenai pajak.

**g. Hasil Program Goods**

```
PS D:\Semester 3\PBO Praktek\Task 5 Interface>
if ($?) { javac GoodsTest.java } ; if ($?) { java GoodsTest }
Description: apple
Price: 2.0
Calories: 95

Description: Lego
Price: 50.0
Minimum Age: 6
Tax: 3.0

Description: Java Programming
Price: 30.0
Author: John
Tax: 1.7999999999999998
```

## **BAB II**

### **LESSON LEARNED**

Dari studi kasus Employee dan Sortable, saya belajar bagaimana konsep abstract class bekerja dalam pewarisan. Pada Case 1, Sortable bertindak sebagai kontrak yang mewajibkan subclass-nya (Employee) mengimplementasikan method compare, sehingga objek Employee bisa dibandingkan berdasarkan atribut tertentu, dalam hal ini gaji. Hal ini memberi fleksibilitas karena method sorting (shell\_sort) tidak perlu tahu detail internal Employee, cukup memanggil compare. Pada Case 2, saya menemukan keterbatasan Java yang tidak mendukung multiple inheritance, sehingga Manager tidak bisa langsung extends Employee extends Sortable. Namun, saya juga memahami solusinya, yaitu memanfaatkan single inheritance: karena Employee sudah mewarisi Sortable, maka Manager otomatis juga merupakan Sortable. Jika dibutuhkan logika yang berbeda, Manager cukup melakukan override method yang relevan.

Sedangkan dari studi kasus Goods, Food, Toy, dan Book, saya belajar bagaimana kombinasi inheritance dan interface digunakan secara efektif. Goods berperan sebagai parent class yang menyimpan atribut dasar (deskripsi dan harga), sementara subclass seperti Food, Toy, dan Book menambahkan atribut spesifik sesuai jenis barang. Melalui interface Taxable, hanya class tertentu (Toy dan Book) yang diwajibkan menghitung pajak, sehingga kita bisa mengontrol perilaku mana yang harus ada tanpa memaksa semua subclass mewarisinya. Dari sini saya memahami bahwa abstract class lebih cocok dipakai ketika ada perilaku dasar yang harus diturunkan ke semua child, sedangkan interface lebih cocok untuk mendefinisikan perilaku tambahan yang bisa dipilih sesuai kebutuhan. Dengan kedua pendekatan ini, desain program menjadi lebih fleksibel, terstruktur, dan mudah dikembangkan.