**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**

[BAB I HASIL PENGERJAAN 3](#_Toc209640082)

[1. Hasil Pengerjaan Task 3.1 3](#_Toc209640083)

[a. Case 1 3](#_Toc209640084)

[b. Hasil Case 1 4](#_Toc209640085)

[c. Case 2 4](#_Toc209640086)

[d. Hasil Case 2 5](#_Toc209640087)

[2. Hasil Pengerjaan Soal 5 Interface 5](#_Toc209640088)

[a. Interface Taxable 5](#_Toc209640089)

[b. Goods (Parent Class) 6](#_Toc209640090)

[c. Food (Subclass) 7](#_Toc209640091)

[d. Toy (Subclass) 7](#_Toc209640092)

[e. Book (Subclass) 8](#_Toc209640093)

[f. GoodsTest (Main Class) 8](#_Toc209640094)

[g. Hasil Program Goods 9](#_Toc209640095)

[BAB II LESSON LEARNED 10](#_Toc209640096)

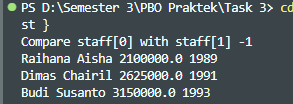
# **BAB I HASIL PENGERJAAN**

## **Hasil Pengerjaan Task 3.1**

### **Case 1**

|  |  |
| --- | --- |
| **Program** | **Penjelasan** |
|  | 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. |
|  | 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. |
|  | 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. |

### **Hasil Case 1**



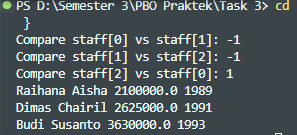
### **Case 2**

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.

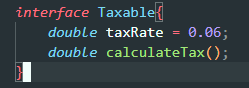
|  |  |
| --- | --- |
| **Program** | **Penjelasan** |
|  | Manager mewarisi semua atribut dan method dari Employee, termasuk compare. Ini memungkinkan Manager langsung bisa dibandingkan dengan Employee. Method raiseSalary di-override untuk menambahkan bonus 0.5% per tahun pengalaman. secretaryName ditambahkan sebagai atribut khusus Manager. |
|  | 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. |

### **Hasil Case 2**



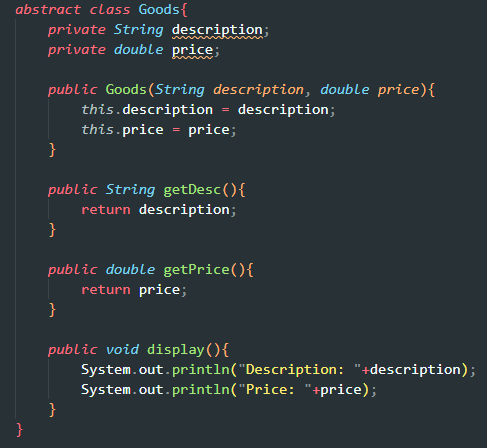
## **Hasil Pengerjaan Soal 5 Interface**

### **Interface Taxable**



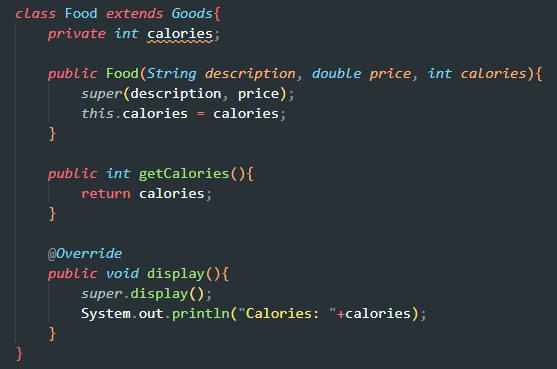
Bagian ini adalah sebuah interface bernama Taxable yang berfungsi sebagai kontrak. Interface ini mendefinisikan sebuah konstanta taxRate dengan nilai 6% dan sebuah method abstrak calculateTax() yang wajib diimplementasikan oleh setiap class yang menggunakan interface ini. Artinya, class yang implements 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.

### **Goods (Parent Class)**



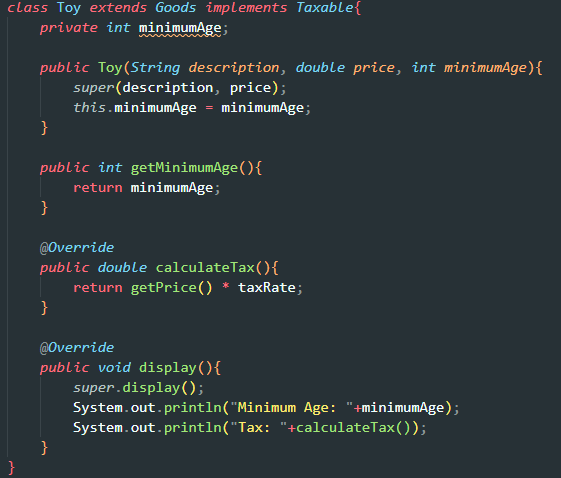
Goods adalah class induk yang menjadi dasar bagi semua barang. Class ini memiliki dua atribut utama yaitu description untuk deskripsi barang, dan price untuk menyimpan harga. Keduanya diset melalui constructor. Tersedia getter (getDescription dan getPrice) untuk mengambil data, serta method display()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

### **Food (Subclass)**



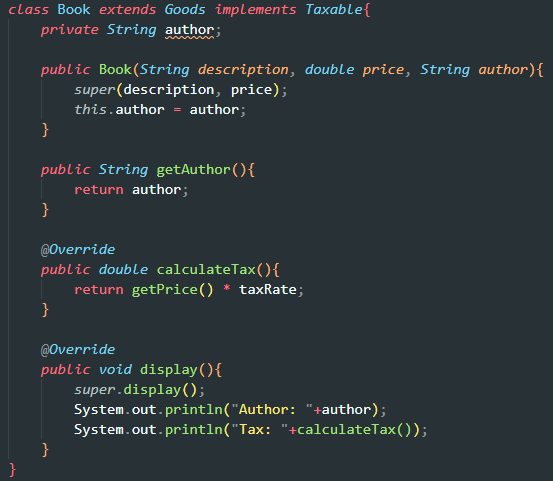
Food adalah subclass dari Goods yang merepresentasikan makanan. Selain atribut description dan price yang diwarisi dari Goods, Food menambahkan atribut khusus yaitu calories untuk menyimpan jumlah kalori makanan. Constructor Food menggunakan super(description, price) untuk memanggil constructor Goods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

### **Toy (Subclass)**



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.

### **Book (Subclass)**



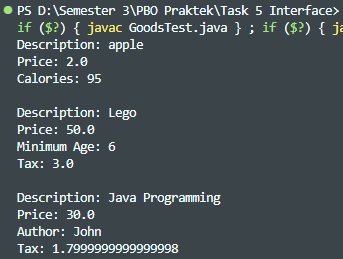
Book adalah subclass lain dari Goods yang juga implements Taxable, sehingga buku bisa dikenakan pajak. Atribut tambahan yang dimiliki Book adalah author untuk menyimpan nama penulis buku. Sama seperti pada Toy, class ini wajib mengimplementasikan calculateTax() untuk menghitung pajak berdasarkan harga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

### **GoodsTest (Main Class)**



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.

### **Hasil Program Goods**



# **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.