LATIHAN EXCEPTION DENGAN JAVA

Penyusun :

Rizky Satria Gunawan

241511089

2C-D3

PROGRAM STUDI D3 TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

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

BAB I: PENDAHULUAN

1.1 Latar Belakang

Pemrograman Berorientasi Objek (PBO) merupakan paradigma pemrograman yang fundamental dalam pengembangan perangkat lunak modern. Konsep-konsep utamanya, seperti **Inheritance (Pewarisan)**, **Polymorphism (Polimorfisme)**, **Abstract Class**, dan **Interface**, memungkinkan pengembang untuk membangun aplikasi yang terstruktur, fleksibel, dan mudah dikelola. Inheritance memungkinkan penciptaan hierarki kelas di mana kelas anak dapat mewarisi atribut dan metode dari kelas induk, mendorong penggunaan kembali kode (*code reusability*). Polymorphism memungkinkan objek dari kelas yang berbeda untuk merespons pemanggilan metode yang sama dengan cara yang spesifik untuk setiap kelas, menciptakan kode yang dinamis. Sementara itu, Abstract Class dan Interface menyediakan kerangka kerja untuk mendefinisikan perilaku umum yang harus diimplementasikan oleh kelas-kelas turunannya, memastikan konsistensi dan desain yang solid. Praktikum ini dirancang untuk memberikan pemahaman mendalam dan pengalaman praktis dalam mengimplementasikan konsep-konsep inti PBO tersebut melalui tiga studi kasus yang berbeda.

1.2 Tujuan Praktikum

Tujuan dari pelaksanaan praktikum ini dijabarkan dalam tiga studi kasus berikut:

1. **Studi Kasus 1 (Another Type of Employee):** Mengimplementasikan konsep **inheritance** untuk menambahkan tipe karyawan baru, yaitu `Commission`, ke dalam hierarki kelas `StaffMember` yang sudah ada, di mana kelas baru ini merupakan turunan dari kelas `Hourly`.
2. **Studi Kasus 2 (Painting Shapes):** Menerapkan konsep **polymorphism** dan **abstract class** untuk menghitung kebutuhan cat pada berbagai bentuk geometri (Rectangle, Sphere, Cylinder) yang merupakan turunan dari kelas abstrak Shape.
3. **Studi Kasus 3 (Polymorphic Sorting):** Memanfaatkan **polymorphism** dan **interface** **`Comparable`** untuk melakukan proses pengurutan (*sorting*) secara generik terhadap berbagai tipe objek, termasuk `Integer`, `String`, dan objek kustom `Salesperson`.

BAB II: DASAR TEORI

Inheritance (Pewarisan)

Inheritance adalah mekanisme di mana sebuah kelas baru (disebut *subclass* atau kelas anak) dapat mewarisi properti (atribut) dan perilaku (metode) dari kelas yang sudah ada (disebut *superclass* atau kelas induk). Manfaat utamanya adalah *code reusability*, karena kita tidak perlu menulis ulang kode yang sama. Di Java, inheritance diimplementasikan dengan kata kunci `extends`. *Contoh Sintaks:*

Java
```
public class Hourly extends Employee {
    // ...
```

```
}
```

Pada contoh di atas, kelas `Hourly` mewarisi semua anggota (non-privat) dari kelas `Employee`.

Polymorphism (Polimorfisme)

Polymorphism, berasal dari bahasa Yunani yang berarti "banyak bentuk", adalah kemampuan suatu objek untuk mengambil banyak bentuk. Dalam PBO, ini berarti objek dari kelas yang berbeda dapat merespons panggilan metode yang sama secara berbeda. Ini memungkinkan satu variabel referensi dari tipe *superclass* untuk merujuk pada objek dari berbagai *subclass*-nya. Manfaatnya adalah fleksibilitas dan skalabilitas program. Sebagai contoh, dalam Studi Kasus 1, metode `pay()` dipanggil pada setiap objek dalam array `staffList`. Meskipun pemanggilan metodenya sama, perilaku yang dieksekusi akan berbeda tergantung pada tipe objek aktualnya (misalnya, `Executive` akan menyertakan bonus, sementara `Hourly` menghitung gaji berdasarkan jam kerja). Begitu pula dengan metode `toString()` yang menampilkan detail spesifik untuk setiap jenis karyawan.

Abstract Class

Abstract Class adalah kelas yang tidak dapat diinstansiasi (tidak bisa dibuat objeknya secara langsung). Kelas ini berfungsi sebagai kerangka dasar untuk kelas-kelas turunannya. Abstract class digunakan ketika kita ingin menyediakan implementasi default untuk beberapa metode tetapi memaksa kelas turunan untuk menyediakan implementasi untuk metode lainnya (yang dideklarasikan sebagai *abstract method*). Perbedaannya dengan kelas konkret adalah kelas konkret dapat diinstansiasi dan semua metodenya harus memiliki implementasi. *Contoh Sintaks:*

Java
```
public abstract class Shape {
    public abstract double area(); // abstract method
}
```

Dalam Studi Kasus 2, kelas `Shape` adalah kelas abstrak karena konsep "bentuk" itu sendiri terlalu umum untuk dihitung luasnya tanpa mengetahui bentuk spesifiknya.

Interface Comparable

Interface adalah tipe referensi di Java yang mirip dengan kelas. Ia adalah kumpulan metode abstrak. Sebuah kelas dapat mengimplementasikan satu atau lebih interface. `Comparable` adalah sebuah interface bawaan Java yang digunakan untuk mendefinisikan urutan alami (*natural ordering*) dari objek sebuah kelas. Interface ini hanya memiliki satu metode, yaitu `compareTo()`. Metode `compareTo(Object o)` membandingkan objek saat ini dengan objek yang diberikan sebagai parameter. Metode ini akan mengembalikan:

- Nilai negatif jika objek saat ini lebih kecil dari objek parameter.
- Nol jika kedua objek sama.
- Nilai positif jika objek saat ini lebih besar dari objek parameter. Dengan mengimplementasikan interface ini, objek dari kelas tersebut dapat diurutkan secara otomatis oleh metode seperti `Sorting.insertionSort()`.

BAB III: ANALISIS DAN PEMBAHASAN

3.1 Studi Kasus 1: Another Type of Employee

Deskripsi Masalah

Pada studi kasus ini, tugasnya adalah memperluas hierarki kelas karyawan yang sudah ada. Tujuannya adalah untuk membuat sebuah kelas baru bernama `Commission` yang merepresentasikan seorang karyawan yang dibayar per jam dan juga menerima komisi dari total penjualan. Kelas `Commission` ini akan menjadi turunan dari kelas `Hourly`.

Diagram Kelas

Diagram kelas untuk studi kasus ini dapat digambarkan sebagai berikut:

```
[StaffMember] (Abstract)
     ^
     |
[Employee]
     ^
     |
[Hourly]
     ^
     |
[Commission]
```

Hubungan ini menunjukkan bahwa `Commission` adalah `Hourly`, `Employee`, dan `StaffMember`.

Penjelasan Kode

- **`Commission.java`** Kelas ini merupakan inti dari studi kasus.
    - **Konstruktor:** Konstruktor `Commission` menerima enam parameter. Lima parameter pertama (nama, alamat, dll.) dilewatkan ke konstruktor kelas induk (`Hourly`) menggunakan `super()`. Parameter keenam, `commissionRate`, digunakan untuk menginisialisasi variabel instance di kelas `Commission`.
    - **`addSales()`:** Metode ini bersifat publik dan digunakan untuk menambahkan nilai penjualan ke total penjualan yang telah terakumulasi oleh karyawan.
    - **`pay()`:** Metode ini di-*override* untuk menyediakan logika perhitungan gaji yang spesifik. Pertama, ia memanggil `super.pay()` untuk mendapatkan gaji berdasarkan jam kerja dari kelas `Hourly`. Kemudian, ia menambahkan hasil perhitungan komisi (`commissionRate * totalSales`). Setelah itu, `totalSales` direset kembali ke nol.
    - **`toString()`:** Metode ini juga di-*override* untuk menambahkan informasi total penjualan ke string yang dihasilkan oleh `super.toString()` dari kelas induk.
- **`Staff.java`** Kelas `Staff` dimodifikasi untuk menguji kelas `Commission` yang baru.
    - Ukuran array `staffList` diperbesar menjadi 8 untuk mengakomodasi dua karyawan baru.
    - Dua objek baru dari kelas `Commission` dibuat dan ditambahkan ke dalam `staffList` dengan data yang telah ditentukan (satu dengan tarif $6.25/jam dan komisi 20%, yang lain $9.75/jam dan komisi 15%).

- Jam kerja dan total penjualan untuk kedua karyawan komisi tersebut diatur untuk pengujian perhitungan gaji.

Hasil dan Analisis

Setelah `StaffTest.java` dijalankan, yang pada dasarnya memanggil metode `payday()` dari kelas `Staff`, output yang dihasilkan adalah sebagai berikut:

```
Name: Sam
Address: 123 Main Line
Phone: 555-0469
Social Security Number: 123-45-6789
Paid: 2923.07
---------------------------------
Name: Carla
Address: 456 Off Line
Phone: 555-0101
Social Security Number: 987-65-4321
Paid: 1246.15
---------------------------------
Name: Woody
Address: 789 Off Rocker
Phone: 555-0000
Social Security Number: 010-20-3040
Paid: 1169.23
---------------------------------
Name: Diane
Address: 678 Fifth Ave.
Phone: 555-0690
Social Security Number: 958-47-3625
Current Hours: 40
Paid: 422.0
---------------------------------
Name: Norm
Address: 987 Suds Blvd.
Phone: 555-8374
Thanks!
---------------------------------
Name: Cliff
Address: 321 Duds Lane
Phone: 555-7282
Thanks!
```

```
Name: Alex Morgan
Address: 12 Market Street
Phone: 555-3210
Social Security Number: 112-23-3344
Current Hours: 35
Total Sales: 400.0
Paid: 298.75
---------------------------------
Name: Jordan Riley
Address: 45 Commerce Ave.
Phone: 555-6543
Social Security Number: 223-34-4455
Current Hours: 40
Total Sales: 950.0
Paid: 532.5
---------------------------------
```

Analisis: Output di atas menunjukkan bahwa perhitungan gaji untuk karyawan komisi sudah benar.

- **Alex Morgan:** Gaji per jam (35 jam * $6.25) + Komisi (0.20 * $400) = $218.75 + $80.00 = **$298.75**.
- **Jordan Riley:** Gaji per jam (40 jam * $9.75) + Komisi (0.15 * $950) = $390.00 + $142.50 = **$532.50**. Ini membuktikan bahwa metode `pay()` yang di-*override* pada kelas `Commission` berhasil menggabungkan logika dari kelas induknya (`Hourly`) dengan logika baru yang spesifik untuk komisi, menunjukkan penerapan inheritance yang tepat.

3.2 Studi Kasus 2: Painting Shapes

Deskripsi Masalah

Tugas pada studi kasus ini adalah menghitung jumlah cat yang diperlukan untuk mengecat tiga objek berbeda: sebuah dek (persegi panjang), sebuah bola besar (bola), dan sebuah tangki (silinder). Program ini harus dirancang menggunakan kelas abstrak dan polimorfisme agar metode untuk menghitung kebutuhan cat dapat bekerja untuk segala jenis bentuk.

Diagram Kelas

Diagram kelas untuk studi kasus ini adalah sebagai berikut:

```
[Shape] (Abstract)
   ^
   |
   +--- [Rectangle]
   |
   +--- [Sphere]
   |
   +--- [Cylinder]
```

Kelas `Shape` adalah kelas induk abstrak, dan tiga kelas lainnya adalah kelas turunan yang konkret.

Penjelasan Kode

- **`Shape.java`** Kelas ini didefinisikan sebagai `abstract` karena tidak ada cara untuk menghitung luas "bentuk" generik. Ia memiliki sebuah metode abstrak `area()` yang tidak memiliki implementasi. Ini memaksa setiap kelas turunan yang konkret untuk menyediakan implementasi metode `area()` sesuai dengan rumus geometrinya sendiri.
- **`Rectangle.java`, `Sphere.java`, `Cylinder.java`** Ketiga kelas ini adalah turunan dari `Shape`. Masing-masing mengimplementasikan (*override*) metode `area()`:
  - `Rectangle`: `area()` mengembalikan `length * width`.
  - `Sphere`: `area()` mengembalikan `4 * Math.PI * radius * radius`.
  - `Cylinder`: `area()` mengembalikan `Math.PI * radius * radius * height`. Setiap kelas juga meng-*override* metode `toString()` untuk memberikan representasi string yang informatif.
- **`Paint.java`** Kelas ini memiliki metode `amount(Shape s)`. Parameter `s` bertipe `Shape`, yang merupakan contoh kunci dari polimorfisme. Metode ini dapat menerima objek dari tipe `Shape` atau tipe turunan apa pun (`Rectangle`, `Sphere`, `Cylinder`). Di dalam metode ini, panggilan `s.area()` akan secara dinamis memanggil metode `area()` yang

sesuai dengan objek aktual yang dilewatkan saat runtime. Jumlah cat dihitung dengan membagi luas area yang didapat dengan tingkat cakupan cat (`coverage`).

Hasil dan Analisis

Output dari program `PaintThings.java` adalah sebagai berikut:

```
Computing amount for Rectangle (length 20.0, width 35.0)
Computing amount for Sphere (radius 15.0)
Computing amount for Cylinder (radius 10.0, height 30.0)
Amount of paint for Rectangle (length 20.0, width 35.0): 2.000 gallons
Amount of paint for Sphere (radius 15.0): 8.078 gallons
Amount of paint for Cylinder (radius 10.0, height 30.0): 26.928 gallons
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\FolderTugas6>
```

Analisis: Pesan "Computing amount for..." yang dicetak dari dalam metode `amount()` menunjukkan bahwa satu metode yang sama dipanggil untuk tiga tipe objek yang berbeda.

- **Deck (Rectangle):** Area = 20 * 35 = 700. Kebutuhan cat = 700 / 350 = **2.0 galon**.
- **Big Ball (Sphere):** Area = $4 * \pi * 15^2 \approx 2827.43$. Kebutuhan cat = 2827.43 / 350 ≈ **8.078 galon**.
- **Tank (Cylinder):** Area = $\pi * 10^2 * 30 \approx 9424.78$. Kebutuhan cat = 9424.78 / 350 ≈ **26.928 galon**. Hasil perhitungan ini membuktikan bahwa polimorfisme bekerja dengan sukses. Metode `amount()` dapat berinteraksi dengan objek `Shape` apa pun tanpa perlu mengetahui tipe spesifiknya, memanggil implementasi `area()` yang benar secara otomatis.

3.3 Studi Kasus 3: Polymorphic Sorting

Deskripsi Masalah

Studi kasus ini berfokus pada penggunaan metode pengurutan generik untuk mengurutkan array dari berbagai tipe objek yang berbeda: angka integer, string, dan objek kustom `Salesperson`. Ini menyoroti kekuatan *polymorphic sorting* yang dimungkinkan oleh penggunaan `interface Comparable`.

Penjelasan Kode dan Modifikasi

- **`Numbers.java`** Kode awal dari `Numbers.java` yang diberikan dalam PDF gagal dikompilasi karena mencoba mengurutkan array `int[]`. Tipe data primitif `int` bukanlah objek dan tidak mengimplementasikan interface `Comparable`, sehingga tidak bisa digunakan dengan metode `Sorting.selectionSort(Comparable[] list)`. Masalah ini diatasi dengan mengubah tipe array dari `int[] intList` menjadi `Integer[] intList`. `Integer` adalah kelas pembungkus (*wrapper class*) untuk `int` yang merupakan objek dan sudah mengimplementasikan `Comparable`.
- **`Sorting.java`** Metode `insertionSort` awalnya dirancang untuk mengurutkan data secara menaik (*ascending*). Untuk memenuhi permintaan studi kasus, metode ini dimodifikasi agar mengurutkan secara menurun (*descending*). Modifikasi dilakukan pada kondisi di dalam perulangan `while`:
    - Kondisi asli: `key.compareTo(list[position-1]) < 0`
    - Kondisi diubah menjadi: `key.compareTo(list[position-1]) > 0` Perubahan ini membuat elemen digeser hanya jika `key` (elemen yang sedang diperiksa) lebih besar dari elemen sebelumnya, menghasilkan urutan menurun.

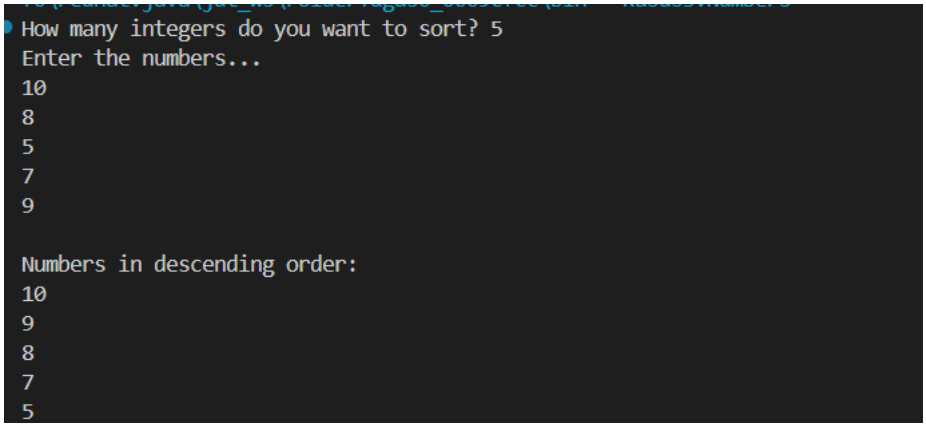
- **`Salesperson.java`** Untuk memungkinkan objek `Salesperson` diurutkan, kelas ini mengimplementasikan `interface Comparable<Salesperson>`. Logika pengurutan kustom didefinisikan di dalam metode `compareTo()`:

    1. **Prioritas Utama:** Perbandingan didasarkan pada `totalSales`. Objek dengan penjualan lebih tinggi dianggap "lebih besar".
    2. **Pemecah Tie (Tie-Breaker):** Jika total penjualan sama, perbandingan dilanjutkan berdasarkan nama belakang (`lastName`), dan jika masih sama, berdasarkan nama depan (`firstName`). Untuk memenuhi permintaan agar urutan nama terbalik (Z-A), hasil dari `compareToIgnoreCase` dinegasikan.

Hasil dan Analisis

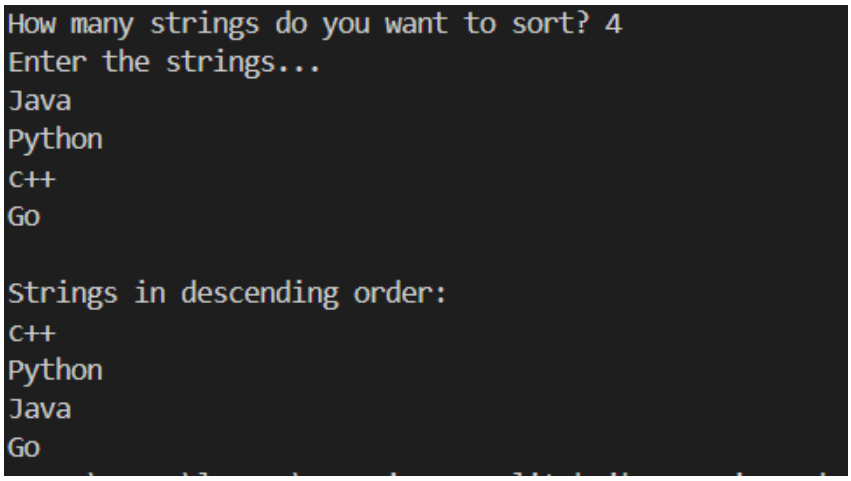
Berikut adalah output dari ketiga program setelah modifikasi:

1. **Pengurutan Angka (Numbers.java):**



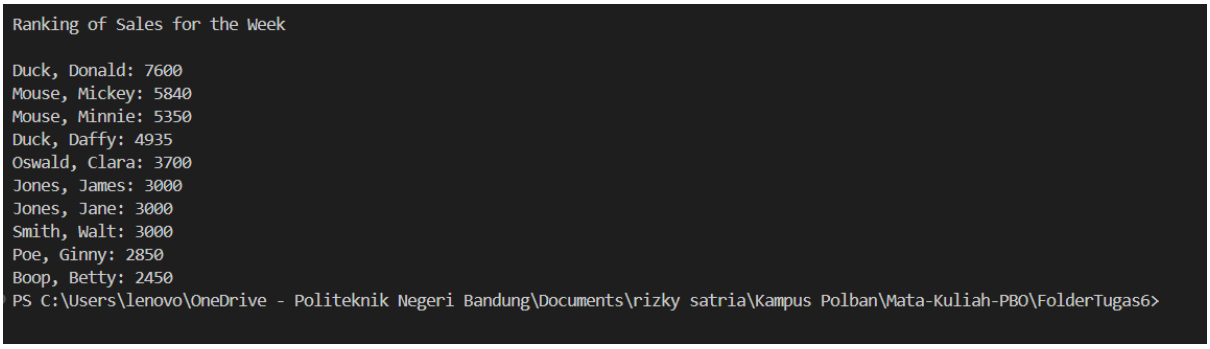
Analisis: Output menunjukkan bahwa angka-angka telah berhasil diurutkan dari yang terbesar hingga terkecil, sesuai dengan modifikasi pada `insertionSort`.

2. **Pengurutan String (Strings.java):**



Analisis: String diurutkan dalam urutan alfabetis terbalik (Z-A), membuktikan bahwa `insertionSort` yang telah dimodifikasi juga berfungsi dengan benar untuk objek `String`.

3. **Pengurutan Salesperson (WeeklySales.java):**



Analisis: Daftar staf penjualan telah diurutkan dengan benar. Prioritas pertama adalah total penjualan dari tertinggi ke terendah. Untuk tiga orang dengan penjualan 3000, mereka diurutkan berdasarkan nama belakang secara terbalik (Smith, Jones, Jones), dan untuk dua Jones, mereka diurutkan berdasarkan nama depan terbalik (Jane, James), sesuai dengan logika dalam metode `compareTo`. Ini menunjukkan kekuatan `interface Comparable` dalam mendefinisikan logika pengurutan yang kompleks untuk objek kustom.

BAB IV: PENUTUP

4.1 Kesimpulan

Praktikum ini telah memberikan pemahaman yang mendalam mengenai konsep-konsep inti Pemrograman Berorientasi Objek. Melalui tiga studi kasus, dapat ditarik beberapa kesimpulan utama:

1. **Inheritance** (Studi Kasus 1) terbukti menjadi alat yang sangat efektif untuk memperluas fungsionalitas sistem tanpa mengubah kode yang sudah ada. Dengan membuat kelas `Commission` sebagai turunan dari `Hourly`, kita dapat mewarisi fungsionalitas yang ada dan menambahkan perilaku baru, yang mendukung prinsip *Open/Closed Principle* (terbuka untuk ekstensi, tertutup untuk modifikasi).
2. **Polymorphism** dan **Abstract Class** (Studi Kasus 2) menunjukkan bagaimana kita dapat menulis kode yang generik dan fleksibel. Metode `amount()` di kelas `Paint` dapat bekerja dengan objek `Shape` apa pun, membuktikan bahwa kita dapat memproses objek dari kelas yang berbeda secara seragam. Ini mengurangi kompleksitas kode dan mempermudah penambahan bentuk-bentuk baru di masa depan.
3. **Interface `Comparable`** (Studi Kasus 3) menyediakan mekanisme standar untuk mendefinisikan urutan objek. Hal ini memungkinkan algoritma pengurutan generik seperti `insertionSort` untuk bekerja pada tipe data apa pun yang mengimplementasikan interface ini, mulai dari tipe bawaan seperti `Integer` dan `String` hingga tipe kustom yang kompleks seperti `Salesperson`.

Secara keseluruhan, praktikum ini menegaskan bahwa kombinasi dari inheritance, polymorphism, abstract class, dan interface merupakan fondasi untuk membangun perangkat lunak yang *reusable*, *maintainable*, dan *scalable*.