**LAPORAN PEMROGRAMAN BERBASIS OBJEK**

**A logo with orange and blue lines

Description automatically generated**

Disusun Oleh:

Dio Rahman Putra - 231511073

**PROGRAM STUDI D3 TEKNIK INFORMATIKA**

**JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA**

**POLITEKNIK NEGERI BANDUNG**

**2024**

1. **Perbedaan antara Checked Exception dan Unchecked Exception**

Jawab:

Checked exceptions dan unchecked exceptions dalam Java memiliki perbedaan penting dalam cara penanganannya dan kapan mereka diperiksa. Checked exceptions adalah jenis exception yang harus ditangani oleh programmer, karena compiler akan memeriksa keberadaan penanganan exception ini saat program dikompilasi. Ini berarti programmer perlu menggunakan blok `try-catch` atau menambahkan `throws` pada metode jika ada kemungkinan terjadi checked exceptions. Checked exceptions biasanya muncul dalam situasi yang dapat diprediksi, tetapi berada di luar kontrol langsung program, seperti ketika terjadi kesalahan saat membaca file atau masalah saat terhubung ke database. Contoh checked exceptions adalah `IOException`, `SQLException`, dan `FileNotFoundException`.

Di sisi lain, unchecked exceptions tidak diperiksa oleh compiler saat program dikompilasi. Exception ini hanya muncul saat program dijalankan. Hal ini berarti programmer tidak perlu menangani unchecked exceptions secara eksplisit, meskipun jika tidak ditangani, exception ini bisa menyebabkan program berhenti tiba-tiba. Unchecked exceptions biasanya terjadi karena kesalahan dalam logika program, seperti mencoba membagi angka dengan nol, mengakses objek yang tidak ada (null), atau mencoba mengakses indeks array yang berada di luar batas. Contoh umum dari unchecked exceptions adalah `ArithmeticException`, `NullPointerException`, dan `ArrayIndexOutOfBoundsException`.

Secara keseluruhan, checked exceptions membantu pengembang untuk menangani kemungkinan kesalahan yang dapat terjadi dalam situasi tertentu, sementara unchecked exceptions biasanya menunjukkan kesalahan yang seharusnya bisa dihindari dengan penulisan kode yang baik. Checked exceptions seringkali berkaitan dengan masalah dari luar program, sedangkan unchecked exceptions lebih berkaitan dengan kesalahan di dalam kode itu sendiri.

|  |
| --- |
| **Soruce Code** |
| Checked exceptions dan unchecked exceptions dalam Java memiliki perbedaan penting dalam cara penanganannya dan kapan mereka diperiksa. \*\*Checked exceptions\*\* adalah jenis exception yang harus ditangani oleh programmer, karena compiler akan memeriksa keberadaan penanganan exception ini saat program dikompilasi. Ini berarti programmer perlu menggunakan blok `try-catch` atau menambahkan `throws` pada metode jika ada kemungkinan terjadi checked exceptions. Checked exceptions biasanya muncul dalam situasi yang dapat diprediksi, tetapi berada di luar kontrol langsung program, seperti ketika terjadi kesalahan saat membaca file atau masalah saat terhubung ke database. Contoh checked exceptions adalah `IOException`, `SQLException`, dan `FileNotFoundException`.  Di sisi lain, \*\*unchecked exceptions\*\* tidak diperiksa oleh compiler saat program dikompilasi. Exception ini hanya muncul saat program dijalankan. Hal ini berarti programmer tidak perlu menangani unchecked exceptions secara eksplisit, meskipun jika tidak ditangani, exception ini bisa menyebabkan program berhenti tiba-tiba. Unchecked exceptions biasanya terjadi karena kesalahan dalam logika program, seperti mencoba membagi angka dengan nol, mengakses objek yang tidak ada (null), atau mencoba mengakses indeks array yang berada di luar batas. Contoh umum dari unchecked exceptions adalah `ArithmeticException`, `NullPointerException`, dan `ArrayIndexOutOfBoundsException`.  Secara keseluruhan, checked exceptions membantu pengembang untuk menangani kemungkinan kesalahan yang dapat terjadi dalam situasi tertentu, sementara unchecked exceptions biasanya menunjukkan kesalahan yang seharusnya bisa dihindari dengan penulisan kode yang baik. Checked exceptions seringkali berkaitan dengan masalah dari luar program, sedangkan unchecked exceptions lebih berkaitan dengan kesalahan di dalam kode itu sendiri. |

|  |
| --- |
| **Solusi** |
| Untuk menyelesaikan studi kasus yang melibatkan konsep pewarisan (inheritance) dan polimorfisme dalam Java, saya menambahkan satu jenis karyawan baru yang diberi komisi pada hierarki kelas yang sudah ada. Karyawan baru ini disebut Commission dan merupakan subclass dari kelas Hourly. Kelas Commission memiliki dua variabel instance tambahan: total penjualan dan komisi. Komisi ini adalah persentase yang dihitung dari total penjualan karyawan, dan variabel tersebut saya tentukan dalam bentuk desimal (misalnya, 0.2 berarti karyawan mendapatkan komisi 20% dari penjualan). Konstruktor untuk kelas Commission menerima enam parameter, lima di antaranya merupakan informasi yang sama seperti pada kelas Hourly (nama, alamat, nomor telepon, nomor jaminan sosial, dan gaji per jam), serta satu parameter tambahan berupa commissionRate.  Selain itu, saya membuat metode `addSales(double totalSales)` untuk menambahkan nilai total penjualan yang dihasilkan oleh karyawan. Lalu, metode `pay()` pada kelas ini saya override untuk menghitung gaji total, yaitu gaji berdasarkan jam kerja ditambah komisi dari penjualan. Setelah gaji dihitung, total penjualan akan direset menjadi 0 untuk periode pembayaran berikutnya. Metode `toString()` juga saya override untuk menampilkan informasi tambahan tentang total penjualan karyawan.  Untuk menguji kelas Commission, saya memperbarui kelas Staff dengan menambahkan dua karyawan baru yang mendapatkan komisi ke dalam array staffList. Karyawan pertama memiliki gaji per jam sebesar $6.25 dan mendapatkan komisi 20% dari total penjualan sebesar $400 setelah bekerja selama 35 jam. Karyawan kedua memiliki gaji per jam sebesar $9.75 dan mendapatkan komisi 15% dari total penjualan sebesar $950 setelah bekerja selama 40 jam. Terakhir, kelas Firm saya gunakan sebagai titik awal untuk menjalankan program, di mana metode `payday()` pada kelas Staff akan mencetak informasi tentang setiap karyawan dan gaji mereka. Pada akhirnya, program ini akan menampilkan daftar semua staf beserta gaji yang dibayarkan, termasuk karyawan yang mendapatkan komisi. |

|  |
| --- |
| **Output Code** |
|  |

1. **Task 2**

Link Github :

<https://github.com/dio-rahman/Pertemuan8-PBOpraktek/tree/3e9f1596e5f36c43b363aa4087918d62fce50149/Task-2-polymorphism>

|  |
| --- |
| **Source Code** |
|  |

|  |
| --- |
| **Solusi** |
| Dalam menyelesaikan studi kasus tentang hierarki kelas bentuk dan perhitungan jumlah cat yang dibutuhkan untuk mengecat berbagai objek, saya mulai dengan mendefinisikan kelas abstrak bernama Shape. Kelas ini memiliki variabel instance yang menyimpan nama bentuk dan metode abstrak untuk menghitung area. Saya memastikan bahwa kelas Shape tidak dapat diinstansiasi langsung dan bahwa setiap bentuk yang diturunkan darinya harus mengimplementasikan metode area. Selanjutnya, saya membuat tiga kelas turunan: Sphere, Rectangle, dan Cylinder. Kelas Sphere menghitung luas permukaan bola menggunakan rumus 4\*PI\*radius^2, sedangkan kelas Rectangle menghitung luas persegi panjang dengan mengalikan panjang dan lebar. Untuk kelas Cylinder, saya menghitung luas permukaan silinder dengan rumus PI\*radius^2\*height. Setelah itu, saya mendefinisikan kelas Paint yang menghitung jumlah cat yang dibutuhkan berdasarkan area bentuk dibagi dengan coverage cat yang diberikan. Terakhir, saya menyusun kelas PaintThings untuk menginstansiasi objek cat dan berbagai bentuk, serta menghitung dan menampilkan jumlah cat yang diperlukan untuk setiap bentuk. Dengan cara ini, saya dapat melihat bagaimana polimorfisme bekerja saat metode amount() memproses berbagai jenis objek Shape dan menghasilkan informasi yang relevan tentang jumlah cat yang diperlukan untuk mengecat masing-masing objek. |

|  |
| --- |
| **Output Code** |
|  |

1. **Task 3**

Link Github :

<https://github.com/dio-rahman/Pertemuan8-PBOpraktek/tree/3e9f1596e5f36c43b363aa4087918d62fce50149/Task-3-polymorphism>

|  |
| --- |
| **Source Code** |
|  |

|  |
| --- |
| **Solusi** |
| Dalam menyelesaikan tugas tentang pengurutan polimorfik, saya memulai dengan menganalisis file Sorting.java yang berisi kelas Sorting dengan algoritma pengurutan selection sort dan insertion sort untuk mengurutkan array objek yang dapat dibandingkan (Comparable). Pertama, saya meninjau file Numbers.java dan menyadari bahwa program ini tidak dapat dikompilasi karena ia mencoba menggunakan tipe data primitif (int) untuk melakukan pengurutan, sedangkan metode dalam Sorting mengharuskan objek yang dapat dibandingkan. Untuk mengatasi masalah ini, saya melakukan perubahan kecil pada program dengan memanfaatkan fitur autoboxing yang ada di Java 1.5, sehingga konversi dari int ke Integer dapat dilakukan secara otomatis.  Setelah berhasil mengubah Numbers.java, saya kemudian menulis program baru bernama Strings.java yang mirip dengan Numbers.java, tetapi berfokus pada pengurutan array objek String. Saya menyalin dan mengedit Numbers.java untuk menyertakan pengolahan array String dan memastikan bahwa metode pengurutan bekerja dengan baik.  Langkah selanjutnya adalah memodifikasi algoritma insertionSort dalam kelas Sorting agar dapat mengurutkan data dalam urutan menurun, bukan urutan menaik. Saya kemudian memperbarui baik Numbers.java maupun Strings.java untuk memanggil insertionSort alih-alih selectionSort. Setelah melakukan perubahan ini, saya menjalankan kedua program dan memastikan bahwa hasil pengurutannya sudah benar.  Setelah itu, saya melanjutkan dengan menyelesaikan kelas Salesperson yang didefinisikan dalam file Salesperson.java. Saya mengimplementasikan metode compareTo yang membandingkan objek berdasarkan total penjualan. Jika total penjualan objek yang sedang dieksekusi lebih sedikit dibandingkan objek lain, metode ini akan mengembalikan angka negatif; jika lebih banyak, akan mengembalikan angka positif. Untuk menyelesaikan perbandingan, saya juga menambahkan logika untuk mengurutkan nama salesperson secara alfabet jika total penjualannya sama.  Kemudian, saya menggunakan file WeeklySales.java sebagai penggerak untuk menguji metode compareTo dan pengurutan yang telah saya implementasikan. Saya mengkompilasi dan menjalankan program ini untuk memastikan bahwa metode compareTo bekerja dengan benar. Hasilnya, staf penjualan ditampilkan dalam urutan dari penjualan terbanyak ke terendah, dengan empat orang yang memiliki jumlah penjualan sama ditampilkan dalam urutan alfabet terbalik.  Sebagai langkah opsional, saya juga mempertimbangkan untuk memodifikasi WeeklySales.java agar data salesperson dibaca dari input daripada ditentukan secara langsung dalam program. Ini akan memberikan fleksibilitas lebih dalam mengelola data dan membuat program lebih dinamis. |

|  |
| --- |
| **Output Code** |
|  |