LAPORAN PRAKTIKUM KE 5

PEMROGRAMAN BERORIENTASI OBJEK

Nama : Malikah Hiabtul Mulkiyyah

Nim : 2403072

Kelas : TI2C

1. Laporan praktikum **Polymorphism** (method overloading & method overridding)

Polymorphism adalah konsep dalam pemrograman berorientasi objek (OOP) yang berarti "banyak bentuk", memungkinkan objek dari kelas yang berbeda untuk merespons pesan yang sama (seperti pemanggilan metode) dengan cara yang unik untuk masing-masing. Ini memberikan fleksibilitas dan penggunaan kembali kode, karena memungkinkan satu antarmuka digunakan untuk berbagai jenis objek, sehingga objek-objek tersebut dapat diperlakukan sebagai objek dari superkelas yang sama namun tetap memiliki implementasi metode yang berbeda.

1. Method Overloading

Method ovrloading terjadi pada sebuah class yang memiliki nama method yang sama tapi memiliki parameter dan tipe data yang berbeda. Tujuan dari method overloading ini ada untuk memudahkan penggunaan atau pemanggilan method dengan fungsionalitas yang mirip. Method overloading memiliki beberapa aturan, yaitu :

* Nama method harus sama dengan method lainnya.
* Parameter haruslah berbeda.
* Return boleh sama, juga boleh berbeda.

1. Method Overridding

Method Overriding terjadi ketika subclass mendefinisikan ulang method yang sudah ada di superclass dengan signature yang sama. Tujuannya agar subclass bisa memberikan implementasi spesifik sesuai kebutuhan, meskipun method dasarnya sama. Method overridding memiliki beberapa ciri diantaranya adalah nama method sama, parameter sama, dan harus ada hubungan inheritance. Berikut beberapa aturan dalam method overridding :

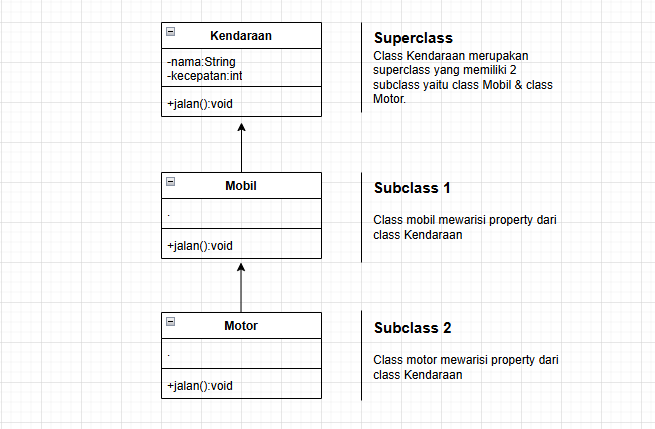
* Mode akses overriding method harus sama atau lebih luas daripada override method.
* Subclass hanya dapat dan boleh meng-override method superclass satu kali saja. Tidak boleh ada lebih dari satu method yang sama pada kelas.
* Soal aturan hak akses, setiap subclass tidak boleh mempunyai hak akses method overriding yang ketat dibandingkan dengan hak akses method pada superclass ataupun parent class.

Agar lebih mudah dipahami kita akan buat contoh untuk masing-masing materi diatas.

1. Contoh untuk Polymorphism

Sebuah aplikasi transportasi memiliki class kendaraan sebagai superclass nya, lalu ada subclass Mobil dan Motor. Semua class memiliki method jalan() tetapi implementasinya berbeda.

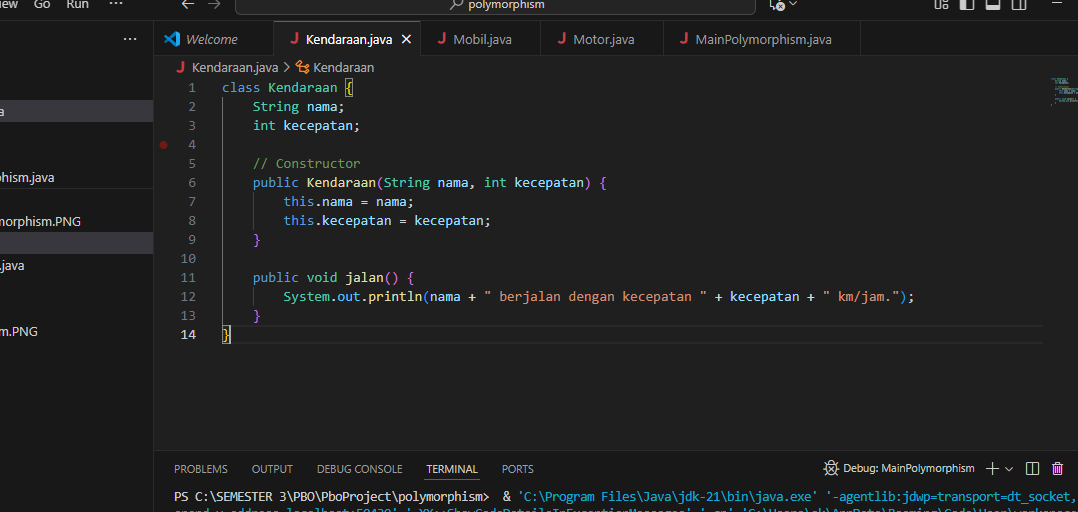
Kita buat dulu class diagram nya :



Kemudian kita buat class Kendaraan, class Mobil, Class Motor, serta class MainPolymorphism :

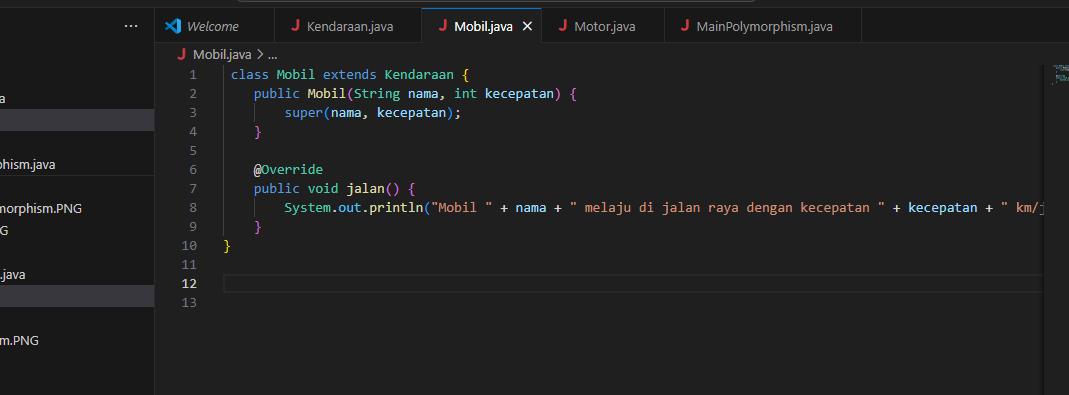
\*Class Kendaraan

Class Kendaraan sebagai superclass akan mewarisi semua property yang dia miliki kepada subclass nya.



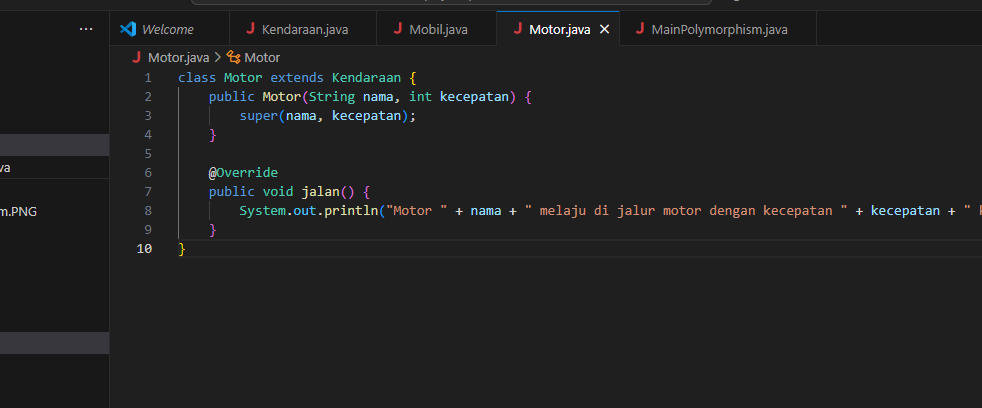
\*Class Mobil

Keyword extends artinya class Mobil mewarisi semua property yang ada pada class Kendaraan.



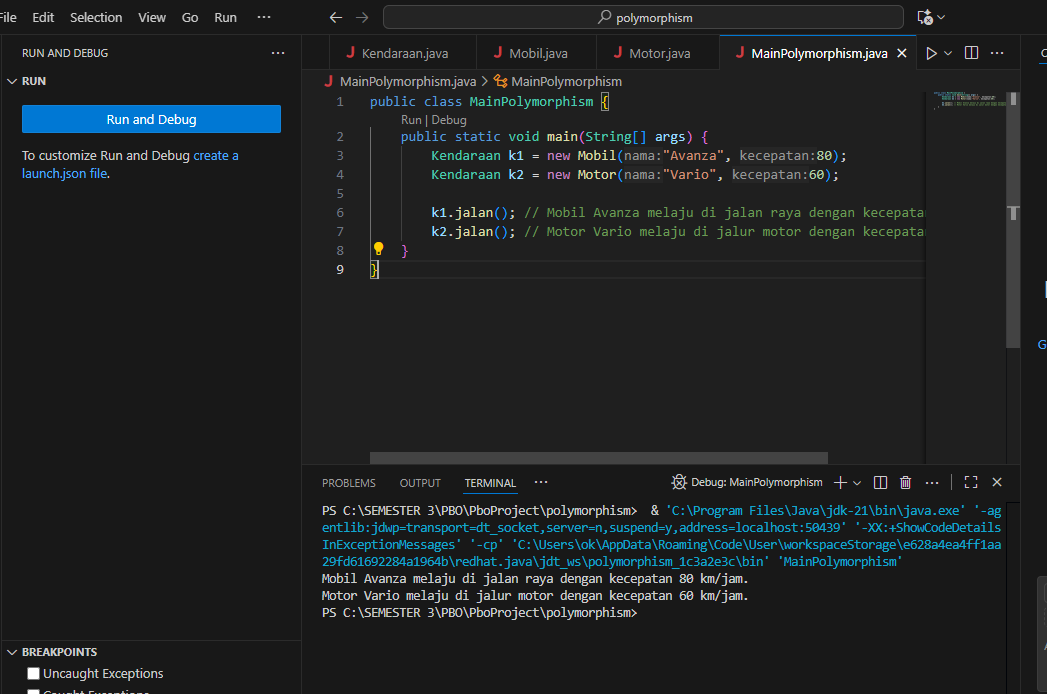
\*Class Motor

Keyword extends pada class Motor artinya, class Motor mewarisi semua property dari class Kendaraan



\*Class MainPolymorphism serta hasil outputnya

Disini kita membuat objek baru dengan nama k1 & k2 pada class Kendaraan.



1. Contoh untuk Method overloading

Bayangkan kita sedang membuat aplikasi transportasi online (seperti Gojek atau Grab).

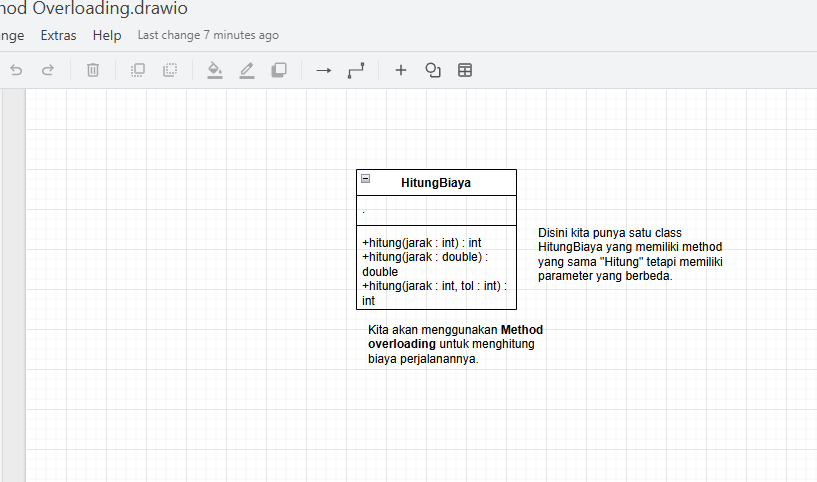
Di aplikasi ini, ada fitur untuk menghitung biaya perjalanan. Tapi, cara menghitung biaya bisa berbeda-beda tergantung input yang diberikan.

Kondisi masalah :

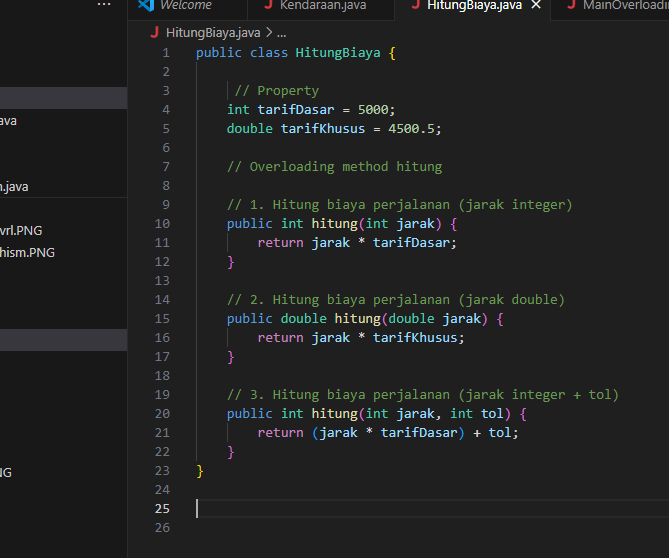
* Jika pengguna hanya memasukkan jarak perjalanan (dalam km, bilangan bulat), maka biaya dihitung dengan tarif Rp 5.000/km.
* Jika pengguna memasukkan jarak dalam **bilangan desimal** (misalnya 12.5 km), maka biaya dihitung dengan tarif khusus **Rp 4.500,5/km**.
* Jika perjalanan melewati jalan tol, maka ada tambahan biaya tol. Jadi, biaya dihitung dengan rumus: (jarak × 5.000) + biayaTol.

Kita buat dulu class diagram nya, kemudian kita buat kode program nya:

\*Class Diagram

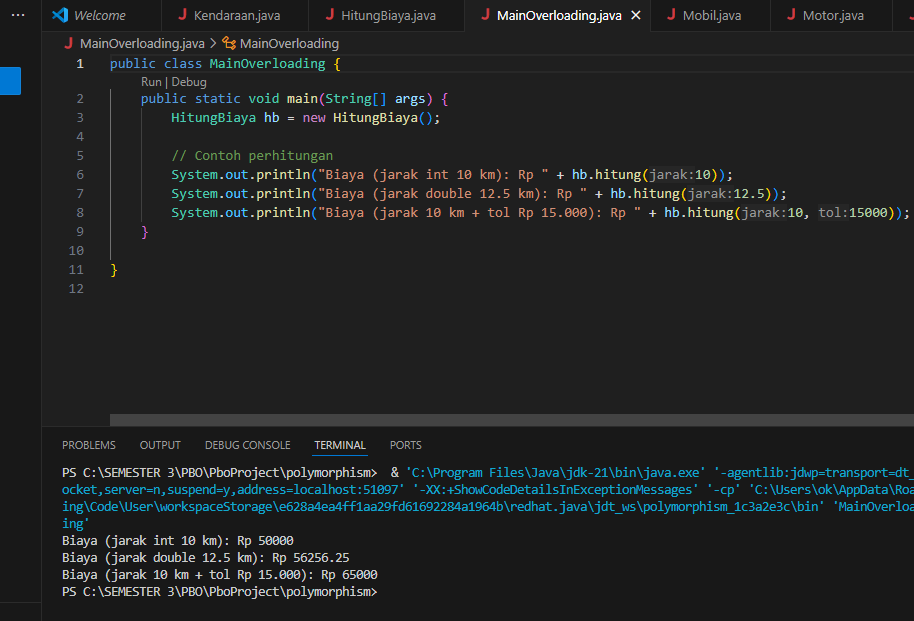


\*Kemudian kita buat class HitungBiaya dan masukkan kode program nya



\*Kemudian buat class MainOverloading untuk menghasilkan output.

Disini kita buat objek baru dengan nama hb pada class HitungBiaya.

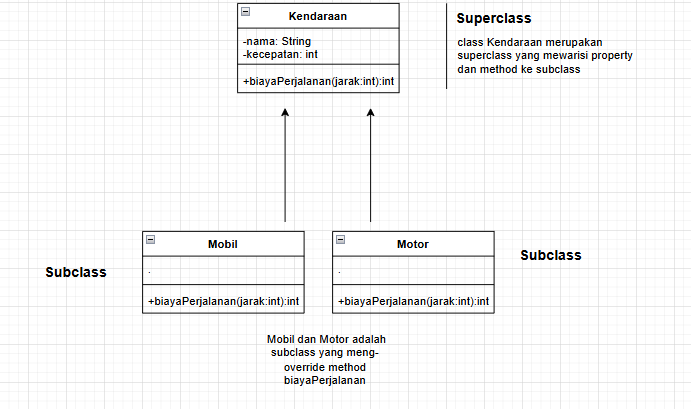


1. Contoh Method Overridding

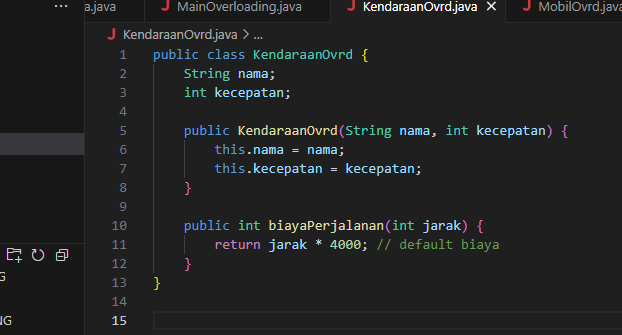
Masih dalam konteks transportasi online, kita punya class induk bernama Kendaraan yang punya method biayaPerjalanan(int jarak). Tetapi tiap kendaraan memiliki tarif yang berbeda :

* Mobil : 5.000 /km
* Motor : 3.000 /km
* Kendaraan (default) : 4.000 /km >> Tipe kendaraan yang belum spesifik (bukan mobil, bukan motor).

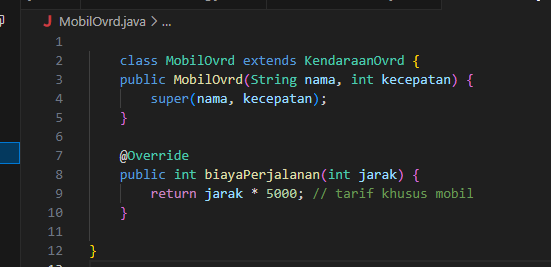
\*Kita Buat class diagram nya



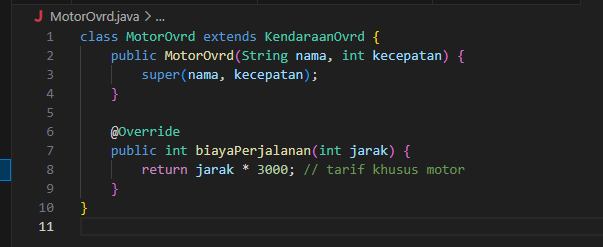
\*Kemudian kita buat class KendaraanOvrd pada kode program



\*Lalu class MobilOvrd

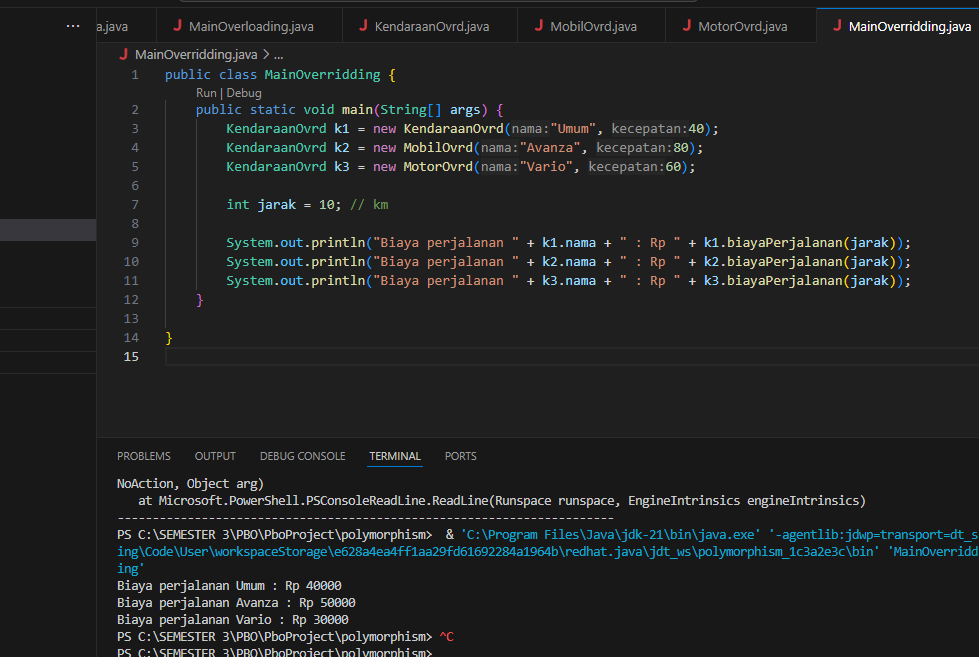


\*Kemudian class MotorOvrd



\* Mobil dan Motor sama-sama menggunakan method biayaPerjalanan() dari superclass Kendaraan. Tapi, mereka mengubah (override) implementasinya sesuai jenis kendaraan. Ketika program dijalankan, Java akan mengeksekusi method yang sesuai dengan objek pada runtime (bukan berdasarkan tipe referensi).

\*Setelah itu kita buat kode program utamanya yaitu MainOverridding.java yang akan kita gunakan untuk runtime.



1. Laporan praktikum **Abstraction (abstract class & interface)**

Abstraksi adalah prinsip inti berorientasi objek untuk menyederhanakan kode dengan menyembunyikan detail implementasi yang rumit dan hanya memperlihatkan fitur-fitur penting. Kelas abstrak menyediakan abstraksi parsial, yang memungkinkan fungsionalitas dan bidang bersama dengan metode abstrak (belum diimplementasikan) dan konkret (diimplementasikan), sementara interface menawarkan abstraksi murni dengan mendefinisikan kontrak metode yang harus diimplementasikan oleh kelas, bertindak sebagai cetak biru untuk perilaku.

1. Abstrak Class

Dapat berisi metode abstrak (tanpa implementasi) dan metode konkret dengan logikanya sendiri. Class yang tidak bisa di-instansiasi secara langsung. Bisa punya method abstract (tanpa body) dan method biasa (dengan body). Digunakan kalau ada relasi is-a dengan shared behavior sebagian atau digunakan untuk membuat kerangka class dengan beberapa implementasi.. Hanya bisa extends 1 abstract class.

1. Interface

Interface adalah kontrak yang berisi sekumpulan method (umumnya tanpa implementasi). Class yang mengimplementasi interface wajib menyediakan implementasi dari method-method tersebut. Interface tidak bisa di-instansiasi langsung. Interface bisa berisi Method abstract, constant (variabel public static final), method default (punya implementasi), method static

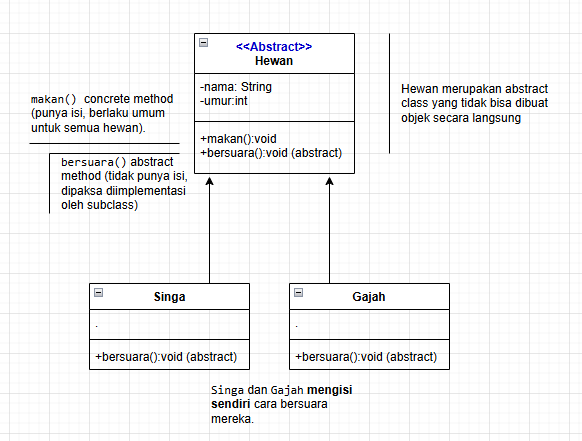
Selanjutnya agar kita lebih memahami materi yang kita bahas, kita buat contoh untuk masing-masing materi yang sudah kita bahas yaitu abstract class & interface.

1. Abstract class

Bayangkan kita membuat aplikasi manajemen hewan di kebun binatang.

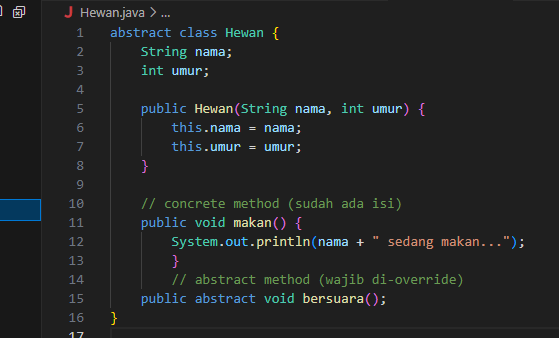
Setiap hewan: Bisa makan() (perilaku umum → sudah ada implementasi). Harus bisa bersuara() (perilaku khusus → beda tiap hewan, jadi harus di-override). Maka kita buat abstract class Hewan sebagai kerangka dasar.

\*Kita buat class diagram nya:

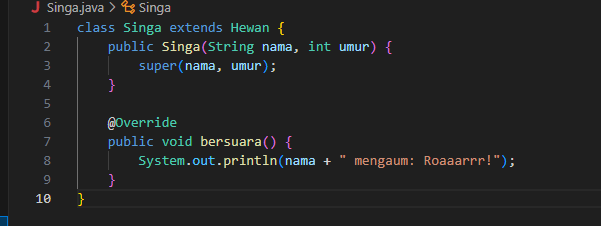


Kemudian kita buat kode program nya :

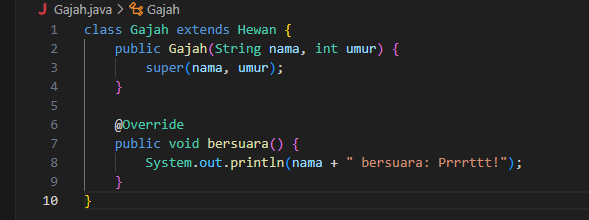
\*Kode program untuk abstract class Hewan



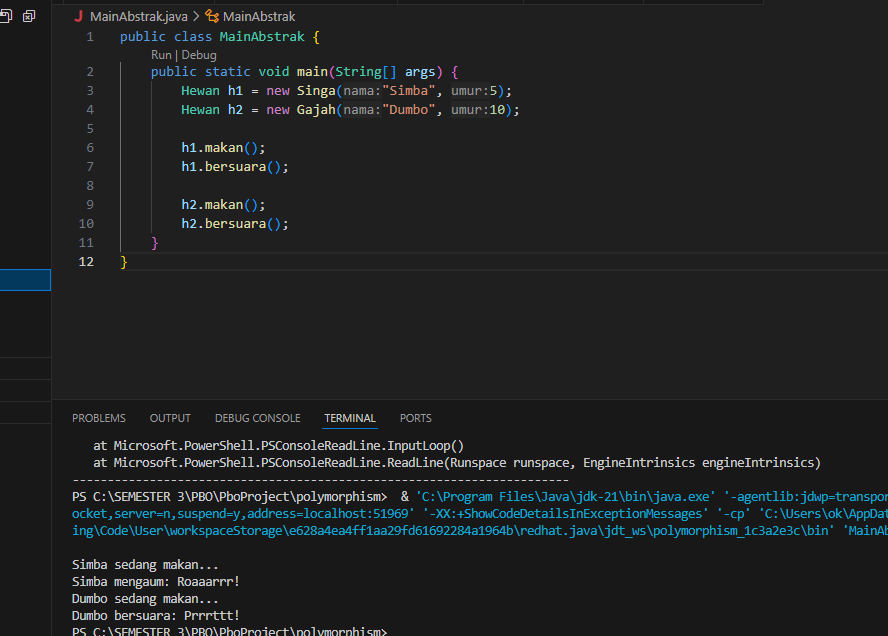
\*Kode program subclass Singa



\*Kode program subclass Gajah



\*Kode program utama yaitu MainAbstract agar bisa melakukan runtime

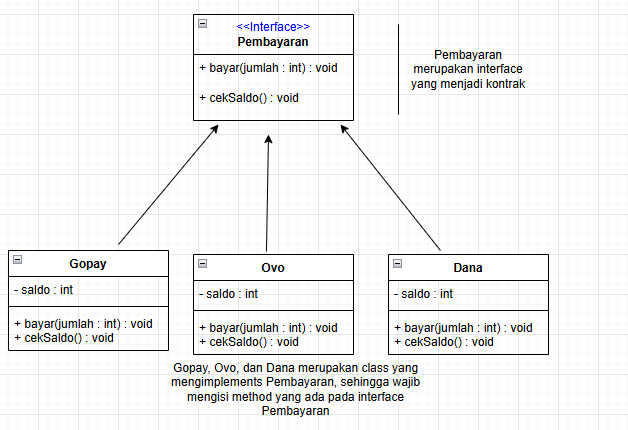


1. Interface

Untuk contoh interface kita akan membuat sebuah aplikasi pembayaran digital seperti dana, ovo, gopay. Masing-masing memiliki kontrak atau aturan yang sama :

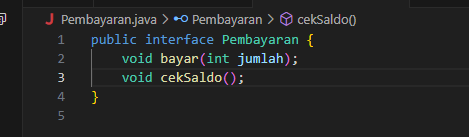
* metode bayar(int jumlah) → untuk membayar sejumlah uang.
* metode cekSaldo() → untuk melihat saldo.

\*Kita buat dulu class diagramnya :

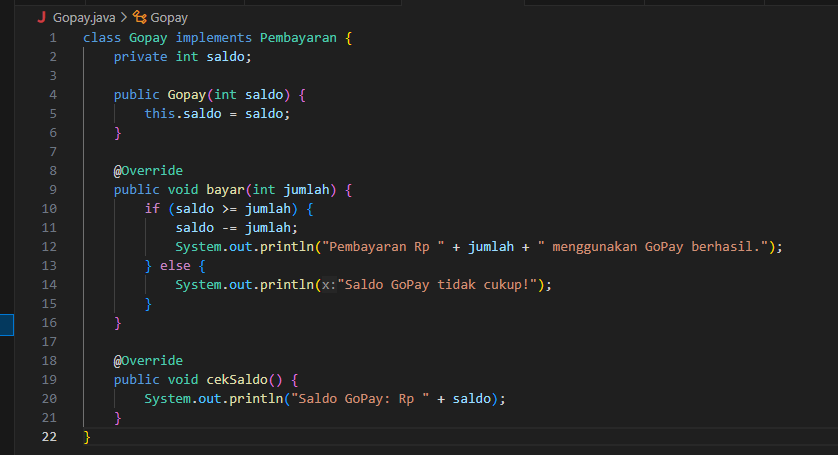


\*Kemudian kita buat kode program untuk masing-masing class dan interface nya

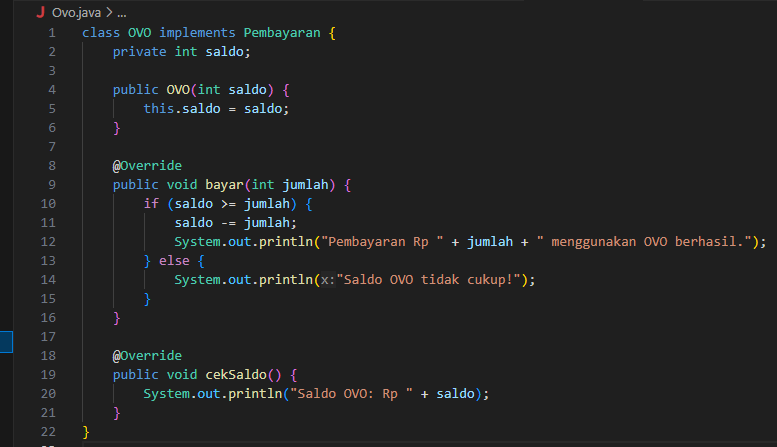
\*Interface Pembayaran



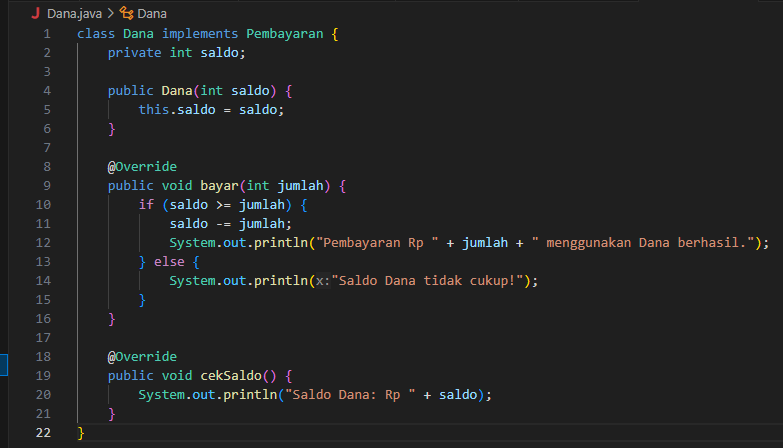
\*Class Gopay



\*Class Ovo



\*Class Dana



\*Kode program MainInterface atau kode program utama untuk melakukan runtime.

