**LAPORAN JOBSHEET 4**

**Praktikum Pemrograman Berbasis Objek**

****

**Disusun Oleh :**

**DHANISA PUTRI MASHILFA**

**NIM. 2341720212**

**TI-2C/08**

**D-IV TEKNIK INFORMATIKA**

**JURUSAN TEKNOLOGI INFORMASI**

**POLITEKNIK NEGERI MALANG**

Jobsheet 04 ‑ Relasi Kelas

# Kompetensi

Setelah menempuh pokok bahasan ini, mahasiswa mampu:

* 1. Memahami konsep relasi kelas;
  2. Mengimplementasikan relasi *has‑a* dalam program.

# Pendahuluan

Pada kasus yang lebih kompleks, dalam suatu sistem akan ditemukan lebih dari satu *class* yang saling memiliki keterkaitan antara *class* satu dengan yang lain. Pada percobaan‑percobaan sebelumnya, mayoritas kasus yang sudah dikerjakan hanya fokus pada satu *class* saja. Pada jobsheet ini akan dilakukan percobaan yang melibatkan beberapa *class* yang saling berelasi.

Misalnya terdapat *class* Laptop yang memiliki atribut berupa merk dan prosesor. Jika diperhatikan lebih rinci, maka atribut prosesor sendiri didalamnya memiliki data berupa merk, nilai *cache* memori, dan nilai *clock*‑nya. Artinya, ada *class* lain yang namanya *Processor* yang memiliki atribut merk, *cache* dan *clock*, dan atribut prosesor yang ada di dalam *class* Laptop itu merupakan objek dari *class* Proceessor tersebut. Sehingga terlihat antara *class* Laptop dan class Processor memiliki relasi (*has‑a*).

|  |
| --- |
| **Laptop** |
| -merk: String  -proc: Processor |
| +Laptop()  +Laptop(merk: String, proc: Processor)  +setMerk(merk: String)  +getMerk(): String  +setProc(proc: Processor)  +getProc(): Processor  +info() |

|  |  |
| --- | --- |
|  | **Processor** |
| -merk: String |
| -cache: double |
| +Processor() |
| +Processor(merk: String, cache: double) |
| +setMerk(merk: String) |
| +getMerk(): String |
|  |
| +setCache(cache: double) |
| +getCache(): double |
| +info() |

Jenis relasi *has‑a* ini yang akan dicontohkan pada percobaan di jobsheet ini. Apabila dilihat lebih rinci lagi, relasi tersebut disebut juga dengan agregasi (*has-a*). Relasi antar kelas yang lain adalah dependensi (*uses-a*) dan *inheritance* (*is-a*). Diperlukan inisiatif mandiri dari tiap mahasiswa untuk memperdalam jenis relasi lain terutama yang tidak dibahas pada mata kuliah ini.

# Praktikum Percobaan 1

1. Perhatikan diagram *class* berikut:

|  |
| --- |
| **Laptop** |
| -merk: String  -proc: Processor |
| +Laptop()  +Laptop(merk: String, proc: Processor)  +setMerk(merk: String)  +getMerk(): String  +setProc(proc: Processor)  +getProc(): Processor  +info() |

|  |  |
| --- | --- |
|  | **Processor** |
| -merk: String |
| -cache: double |
| +Processor() |
| +Processor(merk: String, cache: double) |
| +setMerk(merk: String) |
| +getMerk(): String |
|  |
| +setCache(cache: double) |
| +getCache(): double |
| +info() |

1. Buka *project* baru di *Netbeans* dan buat *package* dengan format berikut:

<identifier>.relasiclass.percobaan1 (ganti <identifier> dengan identitas anda atau nama domain), Contoh: ac.id.polinema, jti.polinema, dan sebagainya).

**Catatan:** Penamaan *package* dengan tambahan identifier untuk menghindari adanya kemungkinan penamaan *class* yang bentrok.

1. Buatlah class Processor dalam *package* tersebut.

# public class Processor {

**}**

1. Tambahkan atribut merk dan *cache* pada class Processor dengan akses modifier

private .

**private String** merk;

**private double** cache;

1. Buatlah *constructor default* untuk class Processor.
2. Buatlah *constructor* untuk class Processor dengan parameter merk dan *cache*.
3. Implementasikan **setter** dan **getter** untuk class Processor.
4. Implementasikan *method* info() seperti berikut:

# public void info() {

System.out.printf("Merk Processor = %s\n", merk); System.out.printf("Cache Memory = %.2f\n", cache);

}

1. Kemudian buatlah class Laptop di dalam package yang telah anda buat.
2. Tambahkan atribut merk dengan tipe String dan proc dengan tipe Object Processor

**private** String merk;

**private** Processor proc;

1. Buatlah *constructor* default untuk *class* Laptop .
2. Buatlah *constructor* untuk *class* Laptop dengan parameter merk dan proc .
3. Selanjutnya implementasikan method info() pada *class* Laptop sebagai berikut

# public void info() {

System.out.println("Merk Laptop = " + merk); proc.info();

}

1. Pada *package* yang sama, buatlah class MainPercobaan1 yang berisi method

main().

1. Deklarasikan Object Processor dengan nama p kemudian instansiasi dengan informasi atribut Intel i5 untuk nilai merk serta 3 untuk nilai *cache* .

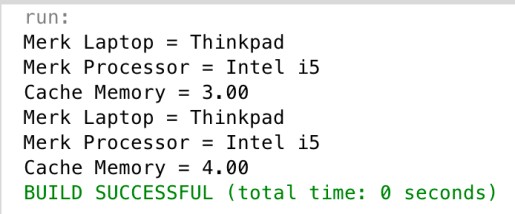
Processor p = **new** Processor("Intel i5", 3);

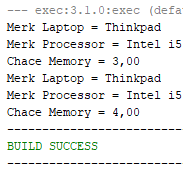
1. Kemudian deklarasikan serta instansiasi Objek Laptop dengan nama L dengan informasi atribut Thinkpad dan Objek Processor yang telah dibuat.
2. Panggil method info() dari Objek L. L.info();
3. Tambahkan baris kode berikut

Processor p1 = **new** Processor(); p1.setMerk("Intel i5"); p1.setCache(4);

Laptop L1 = **new** Laptop(); L1.setMerk("Thinkpad"); L1.setProc(p1); L1.info();

1. *Compile* kemudian *run class* MainPercobaan1, akan didapatkan hasil seperti berikut:





# Pertanyaan

Berdasarkan percobaan 1, jawablah pertanyaan‑pertanyaan yang terkait:

1. Di dalam *class* Processor dan *class* Laptop , terdapat method *setter* dan *getter* untuk masing‑masing atributnya. Apakah gunanya *method setter* dan *getter* tersebut ?

* **Setter** mengubah nilai privat dari luar class
* **Getter** mengambil nilai privat dari luar class

1. Di dalam *class* Processor dan *class* Laptop, masing‑masing terdapat konstruktor default dan konstruktor berparameter. Bagaimanakah beda penggunaan dari kedua jenis konstruktor tersebut ?

* Konstruktor default : tidak memiliki parameter, digunakan untuk objek tanpa nilai awal
* Kostruktor berparameter : memiliki parameter, menginisialisasi atribut dengan nilai yang diberikan saat objek dibuat.

1. Perhatikan *class* Laptop, di antara 2 atribut yang dimiliki (*merk* dan *proc*), atribut manakah yang bertipe *object* ?

* Atribut **proc**

1. Perhatikan *class* Laptop, pada baris manakah yang menunjukan bahwa *class* Laptop

memiliki relasi dengan *class* Processor ?

* 

1. Perhatikan pada *class* Laptop , Apakah guna dari sintaks proc.info() ?

* Sintaks memanggil method **info()** dari objek **Processor** yang diarahlan oleh **proc**

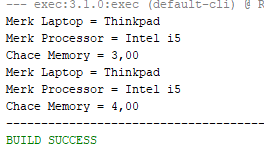
1. Pada *class* MainPercobaan1, terdapat baris kode: Laptop l = new Laptop("Thinkpad", p);. Apakah p tersebut ?

* Object dari class **Peocessor**

Dan apakah yang terjadi jika baris kode tersebut diubah menjadi:

Laptop l = new Laptop("Thinkpad", new Processor("Intel i5", 3));

Bagaimanakah hasil program saat dijalankan, apakah ada perubahan ?



Tidak terjadi perubahan

# Percobaan 2

Perhatikan diagram *class* berikut yang menggambarkan sistem rental mobil. Pelanggan bisa menyewa mobil sekaligus sopir. Biaya sopir dan biaya sewa mobil dihitung per hari.

|  |
| --- |
| **Pelanggan** |
| -nama: String  -mobil: Mobil  -sopir: Sopir  -hari: int |
| +Pelanggan()  +setNama(nama: String)  +getNama(): String  +setMobil(mobil: Mobil)  +getMobil(): Mobil  +setSopir(sopir: Sopir)  +getSopir(): Sopir  +setHari(hari: int)  +getHari(): int  +hitungBiayaTotal(): int |

|  |
| --- |
| **Sopir** |
| -nama: String  -biaya: int |
| +Sopir()  +setNama(nama: String)  +getNama(): String  +setBiaya(biaya: int)  +getBiaya(): int  +hitungBiayaSopir(hari: int): int |

|  |
| --- |
| **Mobil** |
| -nama: String  -biaya: int |
| +Mobil()  +setNama(nama: String)  +getNama(): String  +setBiaya(biaya: int)  +getBiaya(): int  +hitungBiayaMobil(hari: int): int |

* 1. Tambahkan *package* <identifier>.relasiclass.percobaan2.
  2. Buatlah *class* Mobil di dalam *package* tersebut.
  3. Tambahkan atribut *merk* tipe String dan biaya tipe int dengan akses *modifier*

private.

* 1. Tambahkan *constructor default* serta setter dan getter.
  2. Implementasikan method hitungBiayaMobil

**public int** hitungBiayaMobil(**int** hari) {

**return** biaya \* hari;

}

* 1. Tambahkan *class* Sopir dengan atribut nama tipe String dan biaya tipe *int* dengan akses

*modifier* private berikut dengan constructor default.

* 1. Implementasikan method hitungBiayaSopir

**public int** hitungBiayaSopir(**int** hari) {

**return** biaya \* hari;

}

* 1. Tambahkan *class* Pelanggan dengan *constructor default*.
  2. Tambahkan atribut‑atribut dengan akses modifier *private* berikut:

|  |  |
| --- | --- |
| **Atribut** | **Tipe** |
| nama | String |
| mobil | Mobil |
| sopir | Sopir |
| hari | int |

* 1. Implementasikan *setter* dan *getter*.
  2. Tambahkan method hitungBiayaTotal

# public int hitungBiayaTotal() {

**return** mobil.hitungBiayaMobil(hari) + sopir.hitungBiayaSopir(hari);

}

* 1. Buatlah *class* MainPercobaan2 yang berisi method main(). Tambahkan baris kode berikut:

Mobil m = **new** Mobil(); m.setMerk("Avanza"); m.setBiaya(350000); Sopir s = **new** Sopir(); s.setNama("John Doe"); s.setBiaya(200000);

Pelanggan p = **new** Pelanggan(); p.setNama("Jane Doe"); p.setMobil(m);

p.setSopir(s); p.setHari(2);

System.out.println("Biaya Total = " + p.hitungBiayaTotal());

* 1. *Compile* dan jalankan class MainPercobaan2, dan perhatikan hasilnya!



# Pertanyaan

1. Perhatikan *class* Pelanggan. Pada baris program manakah yang menunjukan bahwa *class*

Pelanggan memiliki relasi dengan *class* Mobil dan *class* Sopir ?



1. Perhatikan *method* hitungBiayaSopir pada class Sopir, serta method hitungBiayaMobil pada class Mobil. Mengapa menurut Anda *method* tersebut harus memiliki argument hari ?
2. Perhatikan kode dari *class* Pelanggan. Untuk apakah perintah

mobil.hitungBiayaMobil(hari) dan sopir.hitungBiayaSopir(hari) ?

1. Perhatikan *class* MainPercobaan2. Untuk apakah sintaks p.setMobil(m) dan

p.setSopir(s) ?

1. Perhatikan class MainPercobaan2. Untuk apakah proses p.hitungBiayaTotal()

tersebut ?

1. Perhatikan class MainPercobaan2, coba tambahkan pada baris terakhir dari *method main* dan amati perubahan saat di‑*run*!

System.out.println(p.getMobil().getMerk());

Jadi untuk apakah sintaks p.getMobil().getMerk() yang ada di dalam *method main* tersebut?

# Percobaan 3

Pada percobaan‑percobaan sebelumnya, relasi dalam *class* dinyatakan dalam *one‑to‑one*. Tetapi ada kalanya relasi *class* melibatkan lebih dari satu. Hal ini disebut dengan *multiplicity*. Untuk relasi lebih rinci mengenai *multiplicity*, dapat dilihat pada tabel berikut.

|  |  |
| --- | --- |
| **Multiplicity** | **Keterangan** |
| 0..1 | 0 atau 1 instance |
| 1 | Tepat 1 instance |
| 0..\* | 0 atau lebih instance |
| 1..\* | setidaknya 1 instance |
| n | Tepat n instance (n diganti dengan sebuah angka) |
| m..n | Setidaknya m instance, tetapi tidak lebih dari n |

* 1. Sebuah Kereta Api dioperasikan oleh Masinis serta seorang Asisten Masinis. Baik Masinis maupun Asisten Masinis keduanya merupakan Pegawai PT. Kereta Api Indonesia. Dari ilustrasi cerita tersebut, dapat digambarkan dalam diagram kelas sebagai berikut:

|  |
| --- |
| **KeretaApi** |
| -nama: String  -kelas: String  -masinis: Pegawai  -asisten: Pegawai |
| +KeretaApi(nama: String, kelas: String, masinis: Pegawai)  +KeretaApi(nama: String, kelas: String, masinis: Pegawai, asisten: Pegawai)  +setNama(nama: String)  +getNama(): String  +setKelas(kelas: String)  +getKelas(): String  +setMasinis(masinis: Pegawai)  +getMasinis(): Pegawai  +setAsisten(asisten: Pegawai)  +getAsisten(): Pegawai  +info(): String |

|  |  |
| --- | --- |
|  | **Pegawai** |
| -nip: String |
| -nama: String |
| +Pegawai(nip; String, nama: String) |
| +setNip(nip: String) |
| +getNip(): String |
|  | +setNama(nama: String) |
| +getNama(): String |
| +info(): String |

* 1. Perhatikan dan pahami diagram kelas tersebut, kemudian bukalah IDE anda!
  2. Buatlah *package* <identifier>.relasiclass.percobaan3, kemudian tambahkan

*class* Pegawai.

* 1. Tambahkan atribut‑atribut ke dalam class Pegawai

**private** String nip;

**private** String nama;

* 1. Buatlah *constructor* untuk *class* Pegawai dengan parameter nip dan nama.
  2. Tambahkan *setter* dan *getter* untuk masing‑masing atribut.
  3. Implementasikan *method* info() dengan menuliskan baris kode berikut:

**public** String **info**() { String info = ""; info += "Nip: " + **this**.nip + "\n"; info += "Nama: " + **this**.nama + "\n"; **return** info;

}

* 1. Buatlah *class* KeretaApi berdasarkan diagram *class*.
  2. Tambahkan atribut‑atribut pada *class* KeretaApi berupa nama, kelas, masinis, dan

asisten.

**private** String nama; **private** String kelas; **private** Pegawai masinis; **private** Pegawai asisten;

* 1. Tambahkan *constructor* 3 parameter (nama, kelas, masinis) serta 4 parameter (nama,

kelas, masinis, asisten).

* 1. Tambahkan *setter* dan *getter* untuk atribut‑atribut yang ada pada *class* KeretaApi .
  2. Kemudian implementasikan *method* info()

**public** String **info**() { String info = "";

info += "Nama: " + **this**.nama + "\n"; info += "Kelas: " + **this**.kelas + "\n";

info += "Masinis: " + **this**.masinis.info() + "\n"; info += "Asisten: " + **this**.asisten.info() + "\n"; **return** info;

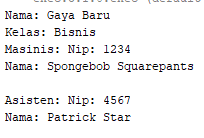
}

* 1. Buatlah sebuah *class* MainPercobaan3 dalam *package* yang sama.
  2. Tambahkan *method* main() kemudian tuliskan baris kode berikut.

Pegawai masinis = **new** Pegawai("1234", "Spongebob Squarepants");

Pegawai asisten = **new** Pegawai("4567", "Patrick Star"); KeretaApi keretaApi = **new** KeretaApi("Gaya Baru", "Bisnis", masinis, asisten);

System.out.println(keretaApi.info());



# Pertanyaan

1. Di dalam *method* info() pada *class* KeretaApi, baris this.masinis.info() dan

this.asisten.info() digunakan untuk apa ?

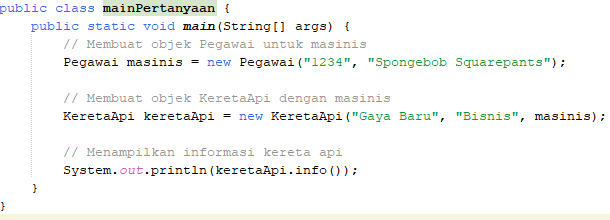
* baris **this.masinis.info()** dan **this.asisten.info()** untuk mengakses informasi dari objek **Pegawai** yang merupakan masinis dan asisten masinis.

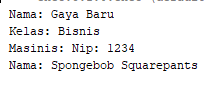
1. Buatlah *main* program baru dengan nama *class* MainPertanyaan pada *package* yang sama. Tambahkan kode berikut pada *method* main() !

Pegawai masinis = **new** Pegawai("1234", "Spongebob Squarepants");

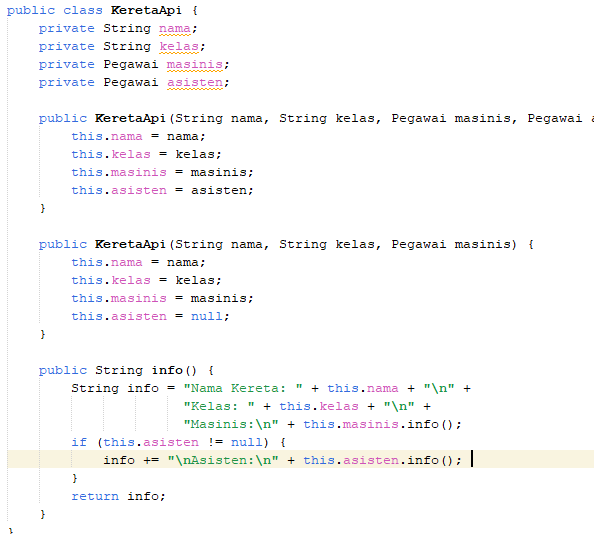
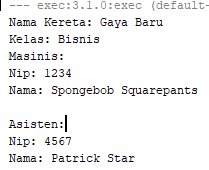
KeretaApi keretaApi = **new** KeretaApi("Gaya Baru", "Bisnis", masinis);

System.out.println(keretaApi.info());



1. Apa hasil output dari *main* program tersebut ? Mengapa hal tersebut dapat terjadi ?

Method **info()** di class KeretaApi memanggil **this,masinin.info(),**  yang mengembalikan String yang berisikan informasi tentang masinis.

1. Perbaiki *class* KeretaApi sehingga program dapat berjalan !

# Percobaan 4

|  |
| --- |
| **Gerbong** |
| -kode: String  -arrayKursi: Kursi[] |
| +Gerbong(kode: String, jumlah: int)  -initKursi()  +setKode(kode: String)  +getKode(): String  +setPenumpang(penumpang: String, nomor: int)  +getArrayKursi(): Kursi[]  +info(): String |

|  |
| --- |
| **Penumpang** |
| -ktp: String  -nama: String |
| +Penumpang(ktp: String, nama: String)  +setKtp(ktp: String)  +getKtp(): String  +setNama(nama: String)  +getNama(): String  +info(): String |

|  |
| --- |
| **Kursi** |
| -nomor: String  -penumpang: Penumpang |
| +Kursi(nomor: String)  +setNomor(nomor: String)  +getNomor(): String  +setPenumpang(penumpang: Penumpang)  +getPenumpang(): Penumpang  +info(): String |

* 1. Perhatikan dan pahami diagram *class* tersebut.
  2. Buatlah masing‑masing *class* Penumpang, Kursi dan Gerbong sesuai rancangan tersebut pada *package* <identifier>.relasiclass.percobaan4.
  3. Tambahkan *method* info() pada *class* Penumpang

**public** String **info**() { String info = "";

info += "Ktp: " + ktp + "\n"; info += "Nama: " + nama + "\n"; **return** info;

}

* 1. Tambahkan *method* info() pada *class* Kursi

**public** String **info**() { String info = "";

info += "Nomor: " + nomor + "\n";

**if** (**this**.penumpang != **null**) {

info += "Penumpang: " + penumpang.info() + "\n";

}

**return** info;

}

* 1. Pada *class* Gerbong buatlah *method* initKursi() dengan akses private.

# private void initKursi() {

**for** (**int** i = 0; i < arrayKursi.length; i++) {

**this**.arrayKursi[i] = **new** Kursi(String.valueOf(i + 1));

}

}

* 1. Panggil *method* initKursi() dalam *constructor* Gerbong sehingga baris kode menjadi berikut:

**public Gerbong**(String kode, **int** jumlah) {

**this**.kode = kode;

**this**.arrayKursi = **new** Kursi[jumlah];

**this**.initKursi();

}

* 1. Tambahkan *method* info() pada *class* Gerbong

**public** String **info**() { String info = "";

info += "Kode: " + kode + "\n";

**for** (Kursi kursi : arrayKursi) { info += kursi.info();

}

**return** info;

}

* 1. Implementasikan *method* untuk memasukkan penumpang sesuai dengan nomor kursi.

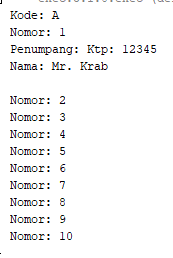
**public void setPenumpang**(Penumpang penumpang, **int** nomor) {

**this**.arrayKursi[nomor - 1].setPenumpang(penumpang);

}

* 1. Buatlah *class* MainPercobaan4 yang berisi *method* main(). Kemudian tambahkan baris berikut!

Penumpang p = **new** Penumpang("12345", "Mr. Krab"); Gerbong gerbong = **new** Gerbong("A", 10); gerbong.setPenumpang(p, 1); System.out.println(gerbong.info());



# Pertanyaan

1. Pada *main* program dalam *class* MainPercobaan4, berapakah jumlah kursi dalam Gerbong A ?



Diatur dengan 10 kursi.

1. Perhatikan potongan kode pada *method* info() dalam *class* Kursi. Apa maksud kode tersebut ?

...

**if** (**this**.penumpang != **null**) {

info += "Penumpang: " + penumpang.info() + "\n";

}

...

Memeriksa atribut **penumpang**  pada objek **kursi**  tidak **null**

1. Mengapa pada *method* setPenumpang() dalam *class* Gerbong, nilai nomor dikurangi dengan angka 1 ?

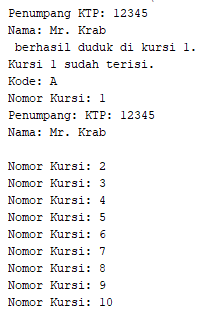
* Menyesuaikan dengan indekes array, karena indeks array dimulai dari 0.

1. Instansiasi objek baru budi dengan tipe Penumpang, kemudian masukkan objek baru tersebut pada gerbong dengan gerbong.setPenumpang(budi, 1). Apakah yang

terjadi ?

* Jika membuat objek baru **budi** dengan tipe **Penumpang** dan memanggil **gerbong.setPenumpang(budi, 1)**, objek **budi** akan menggantikan penumpang yang ada di kursi 1. Jika kursi 1 sudah terisi, penumpang sebelumnya akan dihapus dan diganti dengan **budi.**

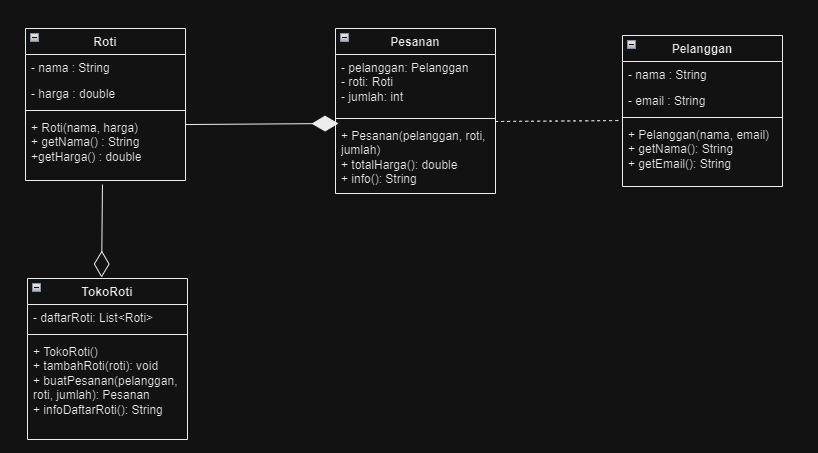
1. Modifikasi program sehingga tidak diperkenankan untuk menduduki kursi yang sudah ada penumpang lain !



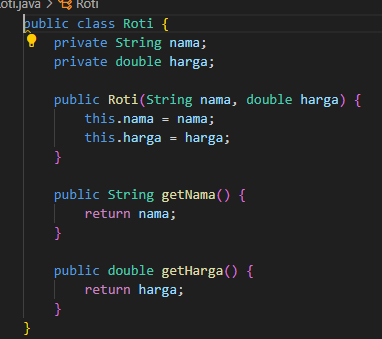
# Tugas

Berdasarkan latihan di pertemuan teori, rancang dengan *class* diagram, kemudian implementasikan ke dalam program! Studi kasus harus mewakili relasi *class* dari percobaan‑percobaan yang telah dilakukan pada materi ini, setidaknya melibatkan minimal 4 *class* (*class* yang berisi *main* tidak dihitung).

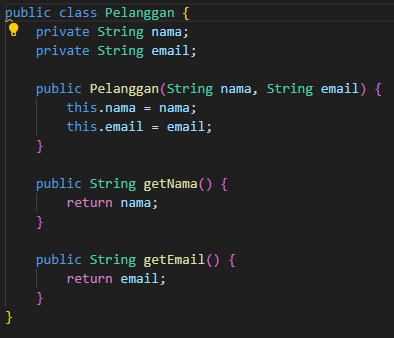
* **Class diagram**



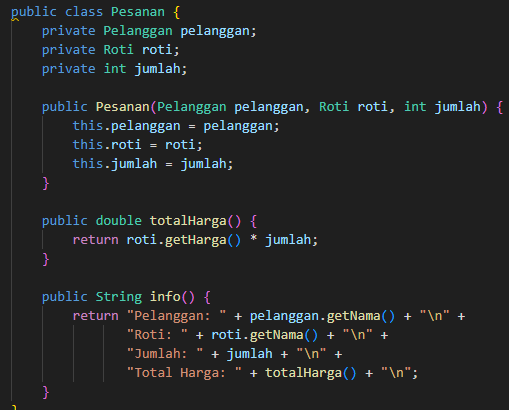
* **Class Roti**



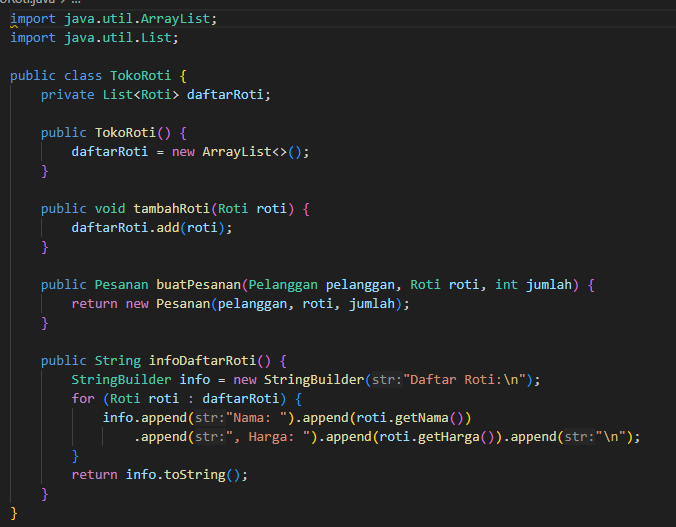
* **Class Pelanggan**



* **Class Pemesanan**



* **Class TokoRoti**



* **Class Main**



* **Output**

