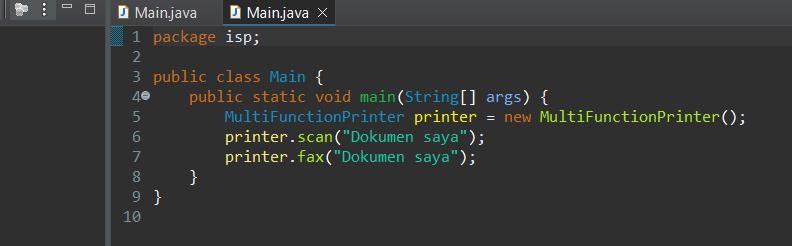
NAMA: Pradana Argo Pangestu

NIM:2311104079/SE-07-02

LAPRAKMODUL6ARSITEKTUR &DESAINPERANGKATLUNAK



# IFax.java

publicinterfaceIFax{

voidfax(Stringdocument);

}

Interface ini hanya mendefinisikan fungsi fax(Stringdocument).

Digunakan untuk objek yang **punya kemampuan fax** saja.

# IScanner.java

public interface IScanner { voidscan(Stringdocument);

}

Interface ini hanya mendefinisikan fungsi scan(String document).

Digunakan untuk objek yang **punya kemampuan scan** saja.

# BasicPrinter.java

publicclassBasicPrinterimplementsIPrinter{ @Override

public void print() { System.out.println("Basicprinting...");

}

}

Catatan: File ini merujuk ke interface IPrinter, tapi kamu belum upload file IPrinter.java. Tapi saya bisa jelaskan berdasarkan konteks

Basic Printer adalah printer sederhana yang **hanyabisaprint**,sesuai dengan prinsip ISP.

Tidak memaksa implementasi metode scan atau fax jika printer tidak memiliki fitur tersebut.

# MultiFunctionPrinter.java

publicclassMultiFunctionPrinterimplementsIFax,IScanner{ @Override

public void scan(String document) { System.out.println("Scanningdocument:"+document);

}

@Override

public void fax(String document) { System.out.println("Faxingdocument:"+document);

}

}

Merupakan **printer multifungsi**. Mengimplementasikanduainterface:

IFax→bisamengirimfax

IScanner→bisamelakukanscan

Tidak mengimplementasikan IPrinter (mungkin secara sengaja karena tidak semuaMFPpunyafungsiprintdalamcontohini,atauprint()akanditambahkan nanti).

# Main.java

publicclassMain{

publicstaticvoidmain(String[]args){

MultiFunctionPrinterprinter=newMultiFunctionPrinter(); printer.scan("Dokumen saya");

printer.fax("Dokumensaya");

}

}

Membuat objek MultiFunctionPrinter

Memanggil dua fungsi:

scan("Dokumensaya")→output:Scanningdocument:Dokumen saya

fax("Dokumensaya")→output:Faxingdocument:Dokumen saya

# Kesimpulan&NilaiKonsep

Program ini merupakan contoh bagus dari **Interface Segregation Principle(ISP)**:

# “Clients should not be forced to dependoninter faces they do not use.”

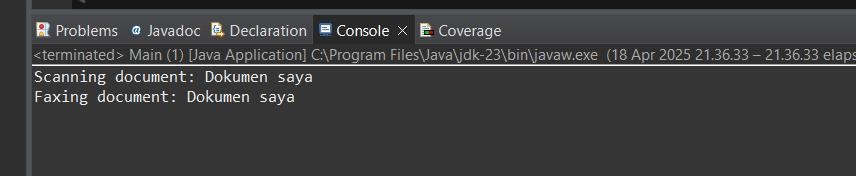
**Baiknya:**

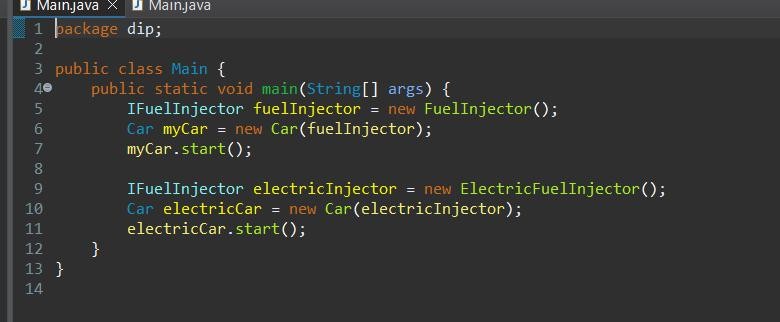
Setiapinterfacehanyamenyimpansatutanggungjawab:IScanner,IFax,dan (kemungkinan besar) IPrinter.

BasicPrintertidakharusimplementscan()ataufax()jikatidak membutuhkannya.

MultiFunctionPrinterbisaimplementfituryangmemangdibutuhkansaja.

# OUTPUT:





1. **IFuelInjector.java(Interface)**

publicinterfaceIFuelInjector{ void inject();

}

**Interface**IFuelInjectormendefinisikansatumetodeinject()yangharus diimplementasikan oleh kelas-kelas injector bahan bakar.

Interfaceinimenjadi**abstraksi** agarkelasCartidaktergantunglangsungpada implementasi tertentu.

# FuelInjector.java(ImplementasiInjectorBensin)

publicclassFuelInjectorimplementsIFuelInjector{ @Override

public void inject() { System.out.println("Bensinfuelinjected!");

}

}

Mengimplementasikan I Fuel Injector.

Saat metode inject()dipanggil ,akan mencetak "Bensin fuel injected!".

# ElectricFuelInjector.java(ImplementasiInjectorListrik)

publicclassElectricFuelInjectorimplementsIFuelInjector{ @Override

public void inject() { System.out.println("Listrikfuelinjected!");

}

}

Implementasi alternatif dari I Fuel Injector,namun untuk **mobil listrik**. Menampilkan pesan "Listrik fuel injected!".

# Car.java(KelasMobil)

publicclassCar{

privateIFuelInjectorinjector;

publicCar(IFuelInjectorinjector){ this.injector = injector;

}

publicvoidstart(){

System.out.println("CarIlhamstarting..."); injector.inject();

}

}

KelasCar**tidakbergantunglangsungpadaimplementasiinjector**,tapi pada interface IFuelInjector.

Artinya,bisadigunakanuntuk**berbagaijenisinjector** tanpamengubahisidari kelas ini.

Didalamstart(),mobilakanmencetak"CarIlhamstarting..."lalumemanggil metode inject() dari injector-nya.

# Main.java(ProgramUtama)

publicclassMain{

public static void main(String[] args) { IFuelInjectorfuelInjector=newFuelInjector(); Car myCar = new Car(fuelInjector); myCar.start();

IFuelInjectorelectricInjector=newElectricFuelInjector(); Car electricCar = new Car(electricInjector); electricCar.start();

}

}

Membuatduamobil:

**Mobilbiasa**denganFuelInjector

**Mobillistrik**denganElectricFuelInjector

Keduanya menggunakan konstruktor Car(I Fuel Injector)dan tetap bisa jalan meskipun implementasi injector-nya beda.

# -Kesimpulan & Nilai Konsep

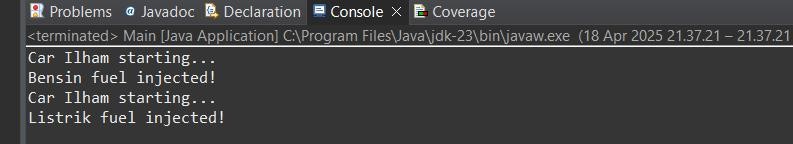
Ini adalah contoh bagus dari **Dependency Inversion Principle**:

Kelas tingkat tinggi(Car)**tidak bergantung langsung** padakelas tingkat rendah (Fuel Injector, Electric Fuel Injector).

Keduanya bergantung pada abstraksi(IFuelInjector).

Kode jadi lebih **fleksibel dan mudah diuji**,karena implementasi injector bisa diganti-ganti tanpa perlu mengubah isi Car.

**OUTPUT:**

****