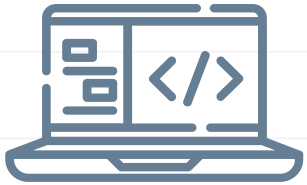
code.id

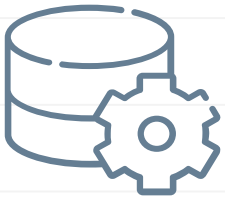
code academy



C#

Object Oriented Programming (OOP)

author : kang dian
Head Trainer



1. Object Oriented Programming (OOP)

OOP adalah metode desain software yang memodelkan karakteristik object di dunia nyata (real wordl) ataupun abstrak menggunakan pendekatan software berupa class dan object, juga interasik antar object.

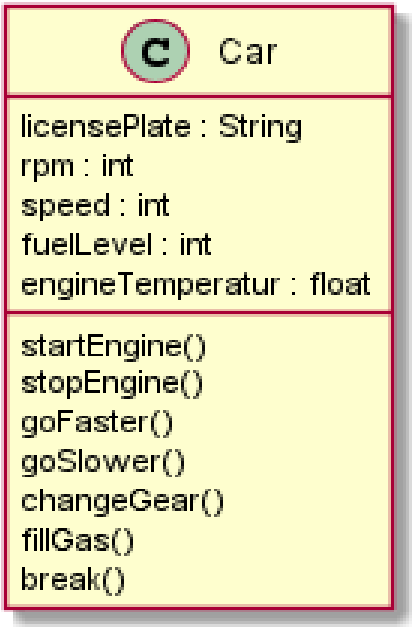
Setiap object di dunia nyata memiliki karakteristik :

- **State** : apa yang dimiliki object.
- **Behaviour** : apa yang bisa dilakukan object.
- **Identity** : apa yang membut object menjadi unik.

Object

State

Behaviour

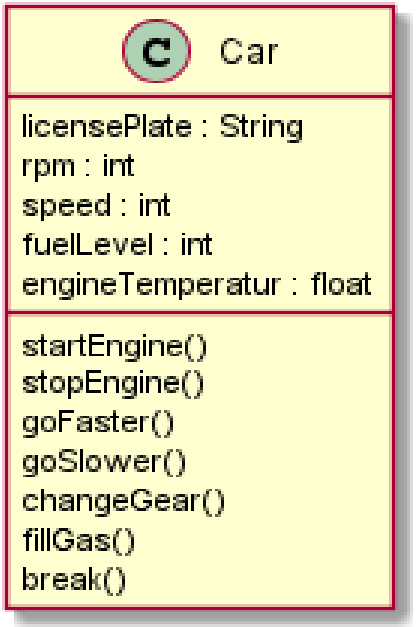


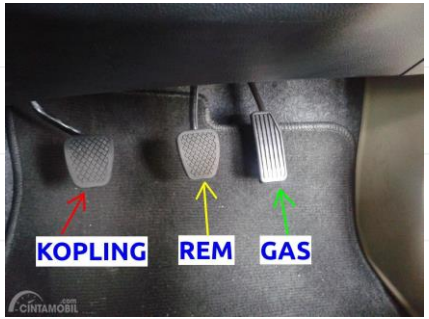
1.1 Example Car Object

Object

State

Behaviour





2. OOP

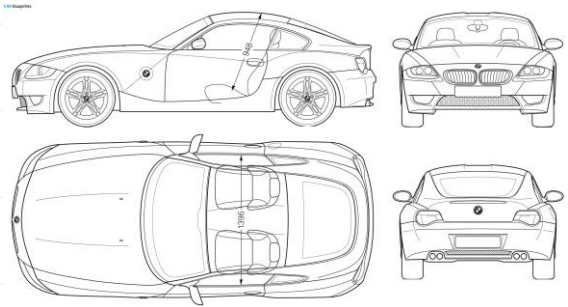
- Encapsulation
- Polymorphism
- Inheritance
- Abstraction

3.1. Class & Object

Blueprint

Car Object (Object Instance)

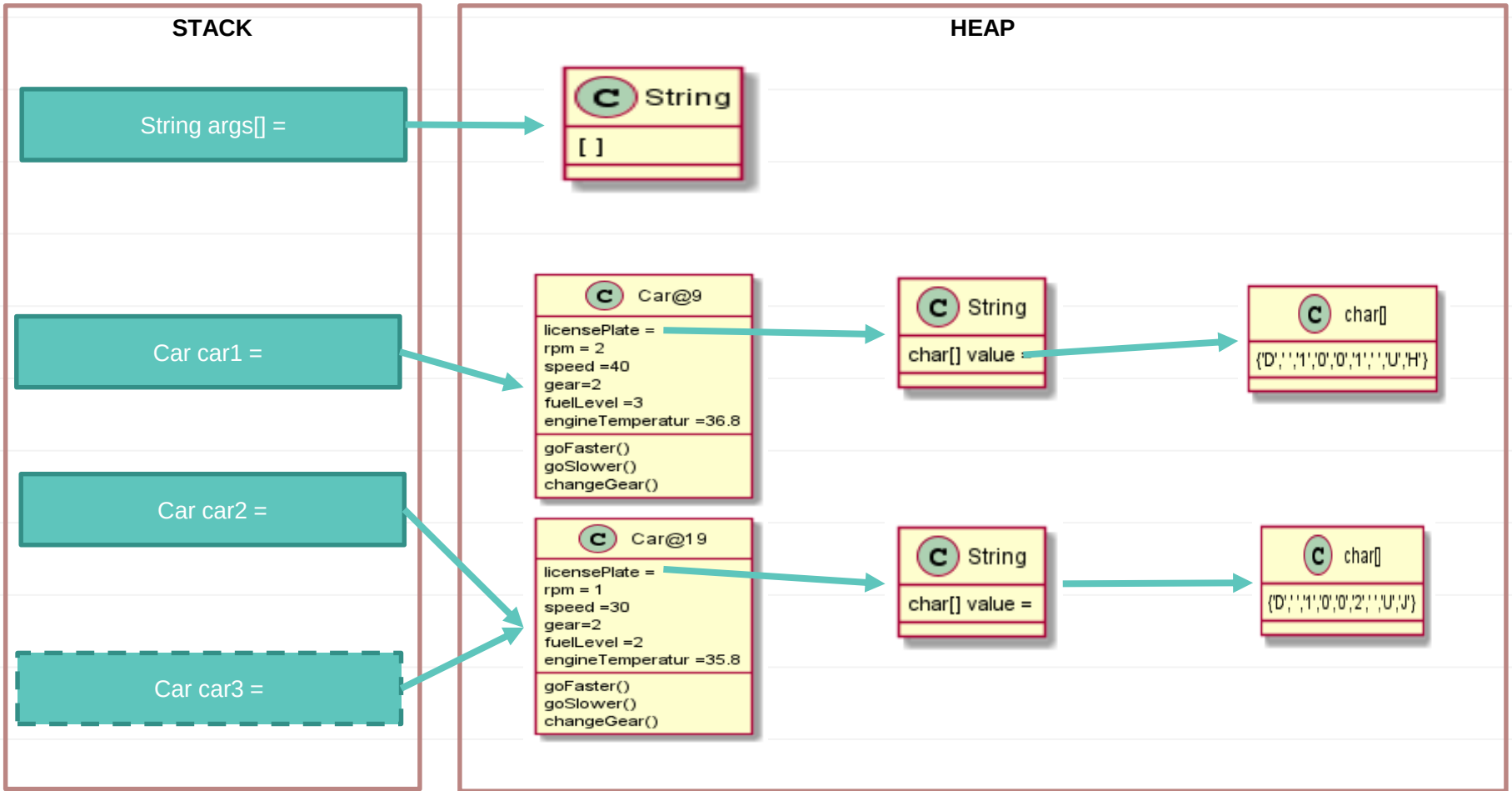
Class Car



4. .NET Memory Model

- .NET memiliki dua memory model yaitu stack memory dan heap memory
- Stack memory menyimpan local variable, sedangkan object disimpan di heap memory
- Satu atau lebih variable bisa memiliki reference ke object yang sama.

4.1. .NET Memory Model



5. Garbage Collection

- .NET tidak support keyword destructor method untuk menghapus object dari memory, sebaliknya java memiliki garbage collection untuk menghapus object di memory.
- .NET secara otomatis akan running garbage collection tiap period, tugas garbage collection :
  - Identifikasi object yang sudah tidak digunakan (no-reference)
  - Deconstruct object object yang no-reference
  - Free up memory yang digunakan oleh object no-reference
  - Defragment memory
- Benefit dari Garbage Collection adalah :
  - Membebaskan programmer dari manage memory, berbeda dengan C++ dimana programmer harus menghapus object secara manual
  - Memastikan integrity program dari memory leak dan mencegah penghapusan object dari memory yang masih digunakan.

7. Static vs Instance Attributes (Data Fields)

```java
public class Car {
   String licensePlate;
   int rpm;
   int speed;
   int gear;
   int fuelLevel;
   double engineTemperature;

  static int totalCar = 0;


}
```

Instance attribute

Static attribute

```java
public class CarClient {
  public static void main(String[] args) {
    // call static atatribute / fields
    Car.totalCar = 4;

    // call instance attribute /fields
    Car car = new Car();
    car.licensePlate = "D 1001 HJ";
  }
}
```

Instance Attribute (Instance Data Fields):

- Dimiliki oleh object instance
- Unique untuk setiap object instance di class yang sama
- Tidak bisa dishare di setiap object instance
- Bisa diakses di method dan constructor menggunakan this.fieldName
- Gunakan this untuk membedakan instance variable atau fieldname dengan local variable.

Static Attribute (Static Data Fields) :

- Unique dalam satu class
- Dimiliki oleh Class bukan object instance
- Dapat dishare oleh semua object instance class yang sama
- Gunakan ClassName.staticFieldName untuk akses value static field
- ClassName hanya optional, hal ini dilakukan jika terdapat variable dengan nama yang sama dengan static field

7.1. Static vs Instance Methods

```
public class Car {
   String licensePlate;
   ...
  static int totalCar = 0;

  // instance method
  public void startEngine() {
    System.out.println("Engine start...");
  }

  // static method
  public static void countCar() {
    totalCar++;
  }
}
```

Instance methods

Static methods

```
public class CarClient {
  public static void main(String[] args) {
    Car car = new Car();
    // call instance method
    car.startEngine();
    // object instance car bisa call static method juga
    car.countCar();

    // call static method
    Car.countCar();

}
```

Static Methods :

- Dimiliki oleh Class bukan object.
- Hanya bisa akses static attribute (static data fields)
- Static method penggunaannya lebih ke pendekatan procedural programming daripada OOP.
- Karena dimiliki oleh Class dan bukan object, maka tidak bisa dihapus oleh Garbage Collection di memory, so please use wisely.
- Pemanggilan method ClassName.staticMethod()

Instance Methods:

- Dimiliki oleh object instance
- Bisa akses static attribute dan instance attribute
- Dipanggil / di invoke hanya ketika sudah dibuat object instance dan diperlukan saja.

7.2. Overloading Methods

```
public class Car {
   String licensePlate;
   int rpm;
   . . .

  static int totalCar = 0;


  // instance method
  public void startEngine() {
    System.out.println("Engine start...");
  }

  public void startEngine(int gear){
    System.out.println("Engine start with gear : "+gear);
  }

  public void countCar(int x) {
    totalCar = totalCar + x;
  }

  // static method
  public static void countCar() {
    totalCar++;
  }
}
```

Overloading Methods

Not Overloading methods, karena satu method menggunakan modifier static

- ***Overloading Methods,*** method memiliki nama yang sama tapi berbeda argument
- Hanya dipanggil oleh object instance

7.2. Variable Argument Length Methods (Varags Method)

```
public class Car {
   String licensePlate;
   int rpm;
   . . .

  public void listPassanger(String... penumpang) {
    for (int i = 0; i < penumpang.length; i++) {
      System.out.println("Penumpang : "+penumpang[i]);
    }
  }

}
```

- Mulai diperkenalkan di java 5
- Argument berfungsi sebagai parameter array
- Jika ada lebih dari satu parameter dalam satu method, tempatkan varags method di akhir.

```
public class CarClient {
  public static void main(String[] args) {
    Car car = new Car();
    car.listPassanger(“Anna",“Eulis");
  }
}
```

Kita bisa input satu
atau lebih value parameter nya

```
//Output :
Penumpang : Anna
Penumpang : Eulis
```

6. Encapsulation

Object tanpa encapsulation tidak lah aman, ibarat kita ubah kecepatan / speed mobil pakai jari tangan langsung sambil injak gas, alhasil informasi tidak sinkron, tidak valid dan tidak aman.

Encapsulation adalah teknik menyembunyikan attribute object dari akses dan assignment langsung. Agar object attribute bisa diakses dan diassignment maka harus menggunakan method.

Benefit encapsulation :

- Request perubahan data attribute bisa divalidasi, sehingga data menjadi valid.
- Menjaga konsistensi attribute object, karena perubahan hanya melalui method
- Melindungi data dari unauthorized access

6.1. Access Modifiers

```
public class Car
{
    private string licensePlate;
    private int speed;
    private int fuelLevel;
    private string brand;
    private string color;

    static int totalCar = 0;

    2 references
    public Car(string licensePlate, int speed, int fuelLevel, string brand, string color)
    {
        this.LicensePlate = licensePlate;
        this.Speed = speed;
        this.FuelLevel = fuelLevel;
        this.Brand = brand;
        this.Color = color;
    }

    3 references
    public string LicensePlate { get => licensePlate; set => licensePlate = value; }
    2 references
    public int Speed { get => speed; set => speed = value; }
    2 references
    public int FuelLevel { get => fuelLevel; set => fuelLevel = value; }
    3 references
    public string Brand { get => brand; set => brand = value; }
    3 references
    public string Color { get => color; set => color = value; }

    1 reference
```

Instance attribute

Static attribute

Access Modifier adalah java keyword yang digunakan untuk mengkontrol access data attributes

Implementasi dapat kita sertakan di :

- Instance & Static Attribute
- Instance & Static Method
- Constructor
- Classes
- Interface

private
Only visible within the same class

public
Visible everywhere

6.2. Getter / Setter

```
public class Car
{
    private string licensePlate;
    private int speed;
    private int fuelLevel;
    private string brand;
    private string color;

    static int totalCar = 0;

    2 references
    public Car(string licensePlate, int speed, int fuelLevel, string brand, string color)
    {
        this.LicensePlate = licensePlate;
        this.Speed = speed;
        this.FuelLevel = fuelLevel;
        this.Brand = brand;
        this.Color = color;
    }

    3 references
    public string LicensePlate { get => licensePlate; set => licensePlate = value; }
    2 references
    public int Speed { get => speed; set => speed = value; }
    2 references
    public int FuelLevel { get => fuelLevel; set => fuelLevel = value; }
    3 references
    public string Brand { get => brand; set => brand = value; }
    3 references
    public string Color { get => color; set => color = value; }

    1 reference
```

- ***Getter (Accessor)*** adalah method yang digunakan untuk akses attributes.
- ***Setter(Mutator)*** adalah method yang digunakan untuk merubah value attributes.
- Getter & Setter dideklarasikan public agar bisa di call diluar class untuk akses dan merubah attributes
- Gunakan modifier private jika attribute hanya ingin dirubah di kelas yang sama dan tidak di public.

7. Constructor

```
public class Car {
  private String licensePlate;
  . . .

  // default constructor
  public Car(){

  }

  // argument constructor
  public Car(String licensePlate, int rpm, int speed, int gear,
  int fuelLevel, double engineTemperature) {
    this.licensePlate = licensePlate;
    this.rpm = rpm;
    this.speed = speed;
    this.gear = gear;
    this.fuelLevel = fuelLevel;
    this.engineTemperature = engineTemperature;
  }

  // argment constructor
  public Car(String licensePlate, int rpm) {
    this.licensePlate = licensePlate;
    this.rpm = rpm;
  }


}
```

Overloading Constructor

- ***Constructor*** adalah special method yang langsung dipanggil ketika compiler meng-create object instance (pada saat di call menggunakan operator new ClassName()).
- Constructor tidak memiliki return type
- Constructor harus memiliki nama yang sama dengan nama Class.
- Jika kita tidak define ekplisit constructor di code, java compiler akan menggunakan default constructor
- Default Constructor tidak memiliki argument
- Menambahkan explicit constructor (constructor argument) akan men-disable default constructor
- Sama seperti method, kita bisa membuat overload constructor, dengan nama yang sama tapi berbeda argument

7.1. How Many Car ?

Kasus berapa banyak total mobil yang dibuat
ketika kita create object instance car

```
public class Car {
   private String licensePlate;
   ...

   static int totalCar = 0;

   public Car(String licensePlate, int fuelLevel) {
     this.licensePlate = licensePlate;
     this.fuelLevel = fuelLevel;
     totalCar++;
   }
}
```

Setiap kali kita create
object instance menggunakan constructor
maka totalCar akan increment

© CanStockPhoto.com - csp63139713

```
public class CarClient {
    Car car1 = new Car("D 1001 UJ", 3);
    Car car2 = new Car("D 1002 UG", 3);
    Car car3 = new Car("D 1003 TR", 3);

    System.out.println(Car.totalCar);
}
```

7.2. Static vs Instance Data Fields

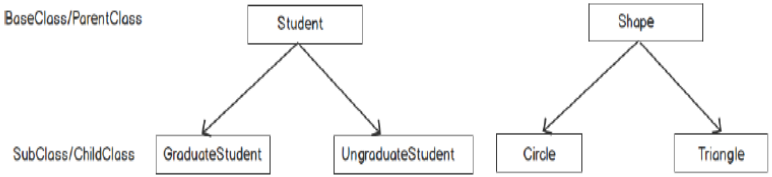
Traffic Light

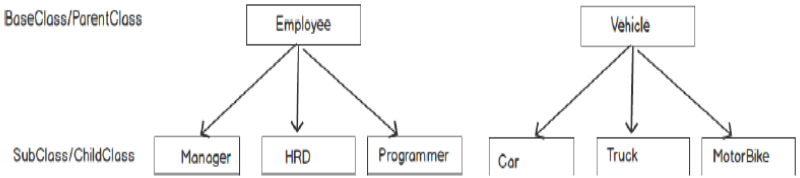
Traffic Light

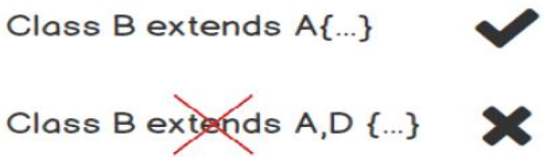
Traffic Light adalah contoh static attribute

Car1, car2, dan car3 adalah object instance yang bisa melihat perubahan attribute static trafficLight

8. Inheritance







- ***Inheritance,*** teknik membuat object dimana attribute dan method nya diambil dari parent class
  - Class yang mewariskan disebut parent-class atau base-class
  - Class yang mewarisi disebut child-class atau subclass
  - Child-Class dapat meng-access attribute dari parent-class
  - Child-Class bisa memiliki attribute atau method yang beda dari parent-class
  - Child-class dapat override method yang dimiliki parent-class
- Inheritance menyerdehanakan pemodelan hirarki dari dunia nyata, sehingga kode script kita lebih sederhana dan re-usable code (digunakan ulang) dari parent-class.
- Inheritance di java hanya dapat di extend dari satu parent class, java tidak support multiple inheritance.
- Java support multi-level inheritance
- Inheritance adalah relasi “is-a” antar object, Car adalah vehicle

8.1. Inheritance

- Java mensupport multiple level inheritance, contoh : Child di extend dari Parent, Parent di extend dari GrandParent.

```
class A {
    String a = null;

    void doA() {
        System.out.println("A says " + a);
    }
}

class B extends A {
    String b = null;

    void doB() {
        System.out.println("B says " + b);
    }
}

class C extends B {
    String c = null;

    void doA() {
        System.out.println("Who cares what A says");
    }

    void doB() {
        System.out.println("Who cares what B says");
    }

    void doC() {
        System.out.println("C says " + a + " " + b + " " + c);
    }
}
```

```
public class ABCDemo {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        a.a = "AAA";
        b.a = "B's A";
        b.b = "BBB";
        c.a = "Who cares";
        c.b = "Whatever";
        c.c = "CCC";
        a.doA();
        b.doB();
        c.doA();
        c.doB();
        c.doC();
    }
}
```

9.5. Polymorphism

Polymorphism adalah kemampuan sebuah object dimana tipe object-nya dapat kita ubah menjadi object lain dengan syarat dua object yang berbeda memiliki relasi parent-child, contoh, object instance Suv dapat kita ubah tipe nya menjadi object Car, karena object Suv extend (mewarisi) dari object Car. Contoh sederhana seperti :

```
Double d = new int (15) // output : 15.0
```

dimana tipe data integer dapat diubah ke Double, atau sebaliknya :

```
int x = (int) 5.0 // output : 5
```

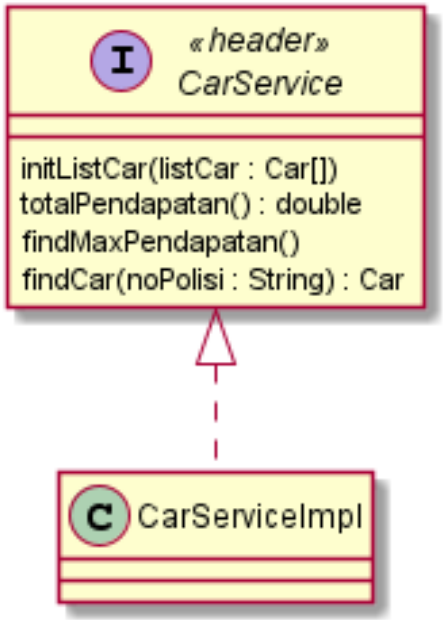
dimana tipe Double bisa kita ubah menjadi integer. Jadi kalo kita implementasi antara Car dan Suv :

```
Car c = new Suv(…);
```

Jika di casting(convert)

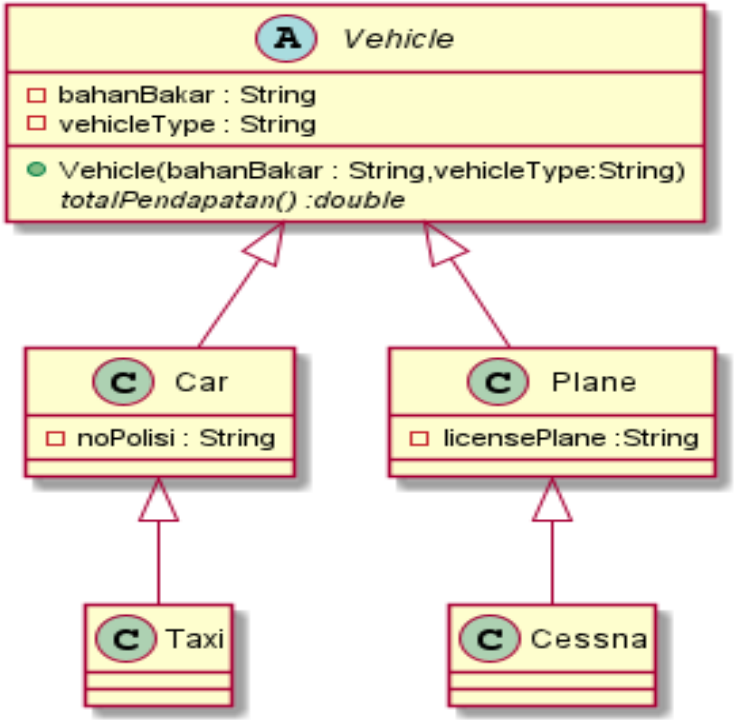
```
Suv  s = (Suv) c;
```

10. Interfaces



- ***Interface*** adalah blueprint untuk class, dalam interface hanya ada sekumpulan method header dan tidak memiliki body implementation.
- Body implementation disimpan di class implementation yang berisi bisnis logic method yang dideklarasikan di header interface.
- Interface menyembunyikan kompleksitas bisnis logic di body implementation
- Interface adalah support multiple implementation

11. Abstract Class



Abstract-Class

Parent-Class

Child-Class

- Abstract Class adalah blueprint atau template untuk superclass(parent class), parent class akan mewarisi attribute dan method dari abstract class. Abstract class masih sama dengan interface, dimana method di implementasikan di subclass. Hanya saja abstact kelas bisa memiliki fields, juga constructor.
- Abstract class tidak bisa di create object instance-nya. Kalo kita coba create object menggunakan new Vehicle(), pasti error.
- Nah supaya bisa di create object instance-nya harus dibuat parent class nya seperti new Car() atau new Plane()

9. Study Case Juragan Mobil

Vehicle Purchase						Income					
NoPolice/NoRegister	VehicleType	Year	Price	Tax(InYear)	Seat	TransactionDate	Rent	Driver	Order	OrderPerKM	Total
D 1001 UM	SUV	2010	350.000.000	3.500.000	4	10/01/2023	500.000	150.000			650.000
D 1002 UM	SUV	2010	350.000.000	3.500.000	4	10/01/2023	500.000	150.000			650.000
D 1003 UM	SUV	2015	350.000.000	3.500.000	5	12/01/2023	500.000	150.000			650.000
D 1004 UM	SUV	2015	350.000.000	3.500.000	5	13/01/2023	500.000	150.000			650.000
D 11 UK	Taxi	2002	175.000.000	1.750.000	4	12/01/2023			45	4.500	202.500
D 12 UK	Taxi	2015	225.000.000	2.250.000	4	13/01/2023			75	4.500	337.500
D 13 UK	Taxi	2020	275.000.000	2.750.000	4	13/01/2023			90	4.500	405.000
ID8089	PrivateJet	2015	150.000.000.000	1.500.000.000	12	23/12/2022	35.000.000	15.000.000			50.000.000
ID8099	PrivateJet	2018	175.000.000.000	1.750.000.000	15	25/12/2022	55.000.000	25.000.000			80.000.000
										SubTotal	133.545.000

Info Summary	
Interface Method	Return Value
GetTotalVehicle()	9
GetTotalVehicle(SUV)	4
GetTotalIncomeVehicle(SUV)	2.600.000
GetTotalIncomeVehicle(TAXI)	945.000
GetTotalIncomeVehicle(PrivateJet)	130.000.000
GetTotalIncomeVehicle()	133.545.000

- Juragan Rental memiliki kendaraan darat dan private jet yang disewakan seperti terlihat di excel.
- Sang juragan ingin tahu berapa total vehicle dan total income baik semua type ataupun tiap tipe nya
- Tugas kamu sebagai programmer:
  - Modelkan class model excel di slide
  - Terapkan encapsulation, inheritance, interface (Info Summary)
  - Report yang disajikan untuk juragan berupa summary dan detailnya.

THANKS!

Any questions?