**LAPORAN PRAKTIKUM**

**JOBSHEET 3**

**MATA KULIAH PBO**

****

**Disusun Oleh :**

**Atsilah Amany Putri Harsuma (2341760011)**

**SIB-2C/08**

**PROGRAM STUDI D4 SISTEM INFORMASI BISNIS**

**JURUSAN TEKNOLOGI INFORMASI**

**POLITEKNIK NEGERI MALANG**

**2024**

# Percobaan

## Percobaan 1 – Tanpa Enkapsulasi

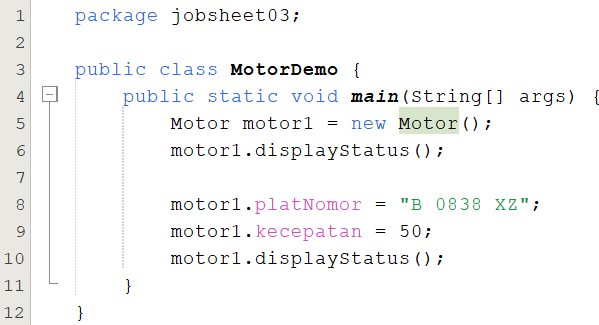
Didalam percobaan enkapsulasi, buatlah class Motor yang memiliki atribut platNomor, isMesinOn (bernilai true jika mesin sedang menyala dan false jika tidak menyala), dan kecepatan serta method displayStatus() untuk menampilkan status motor. UML class diagram class Motor adalah sebagai berikut:

|  |
| --- |
| Motor |
| + platNomor: String  + isMesinOn: Boolean  + kecepatan: int |
| +displayStatus(): void |

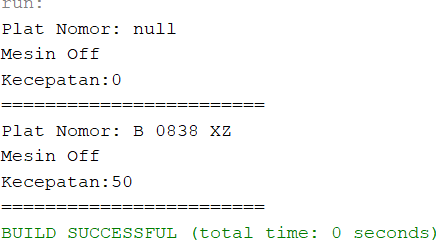
1. Buka Netbeans atau VS code, buat project **Jobsheet03**.
2. Buat class **Motor**. Klik kanan pada package **jobsheet03** – New – Java Class.
3. Ketikkan kode class Motor dibawah ini.



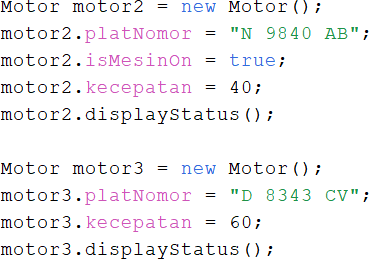
1. Kemudian buat class MotorDemo, ketikkan kode berikut ini.



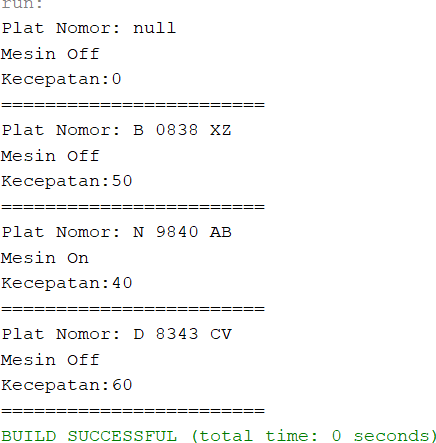
1. Hasilnya adalah sebagai berikut:



1. Selanjutnya buatlah 2 objek motor lagi di class MotorDemo.java



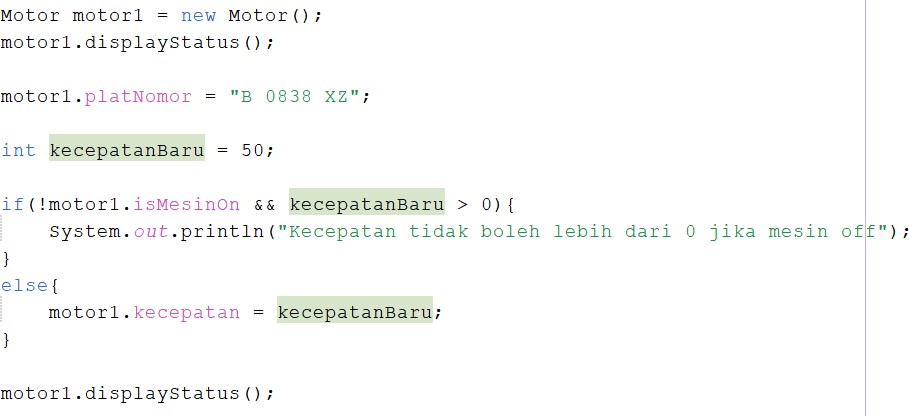
1. Hasilnya sebagai berikut



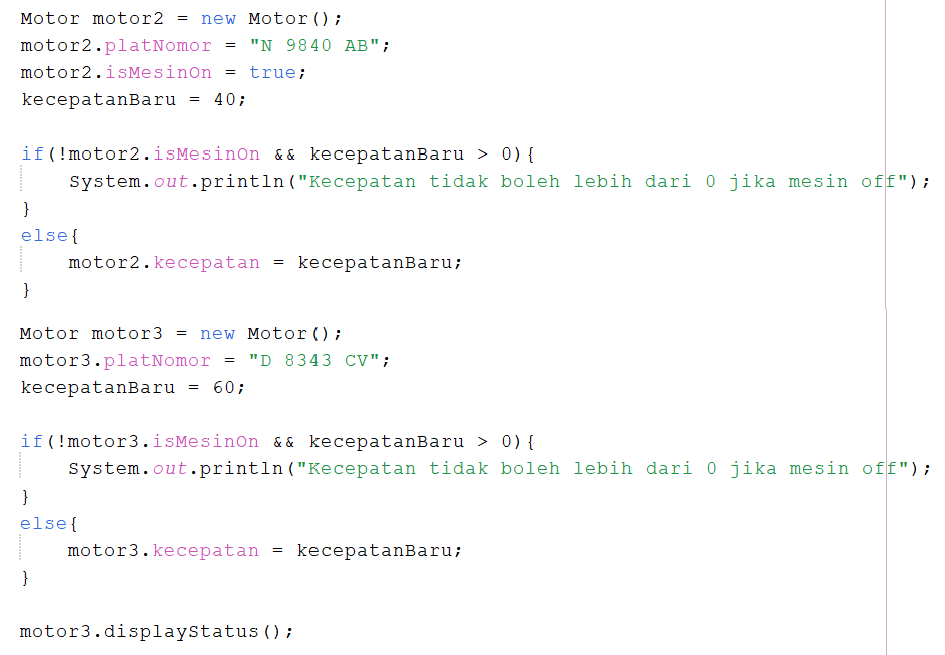
1. Dari hasil di atas, adakah yang janggal?

Pada motor1 dengan plat “B 0838 XZ”, kecepatannya dapat berubah dari 0 ke 50 padahal mesin motor masih dalam kondisi Off. Bagaimana mungkin atribut kecepatan bernilai 50 padahal mesin masih Off? Hal ini karena belum tersedia kontrol/batasan terhadap atribut kecepatan. Padahal, objek di dunia nyata selalu memiliki batasan dan mekanisme bagaimana objek tersebut dapat digunakan. Misalnya motor yang harus dalam keadaan menyala ketika kecepatan lebih dari 0. Kejanggalan ini juga terjadi pada motor ketiga dengan plat nomor "D 8343 CV".

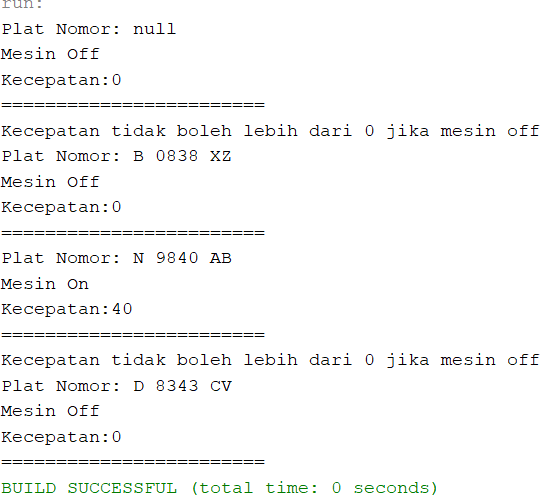
1. Untuk mengatasi hal tersebut, nilai kecepatan baru perlu dicek terlebih dahulu sebelum di- assign ke nilai atribut kecepatan



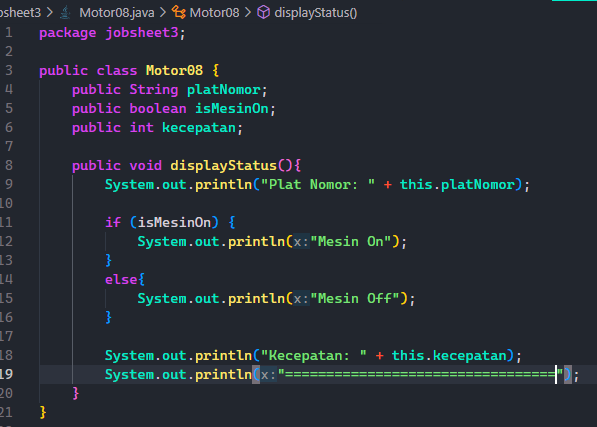
1. Lakukan pengecekan yang sama untuk motor2 dan motor3

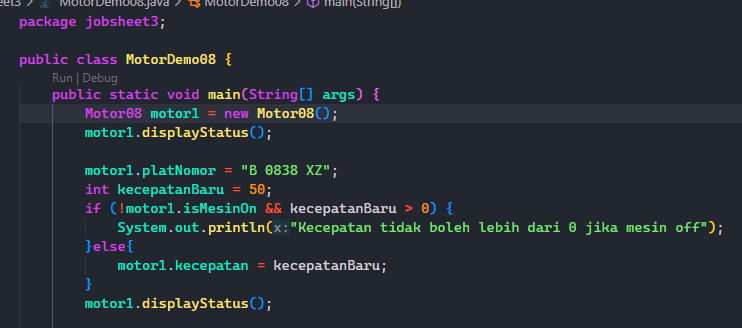


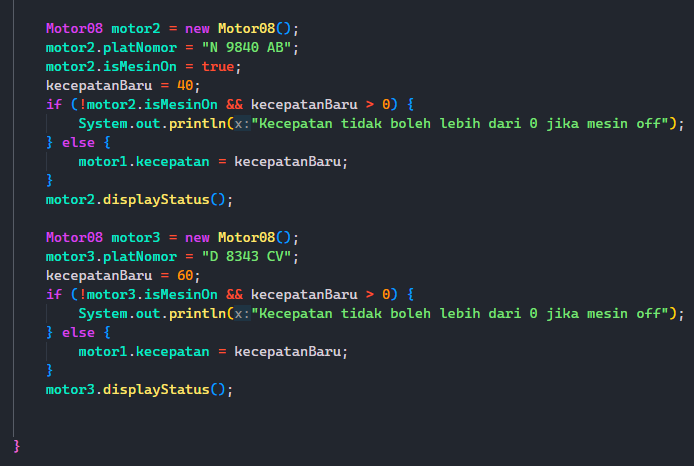
1. Run MotorDemo.java dan perhatikan bahwa sudah terdapat validasi nilai kecepatan terhadap status mesin untuk setiap objek motor



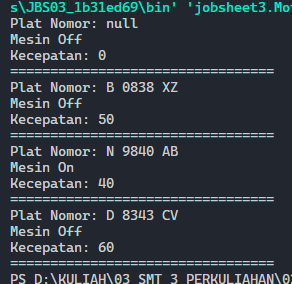
Hasil percobaan:

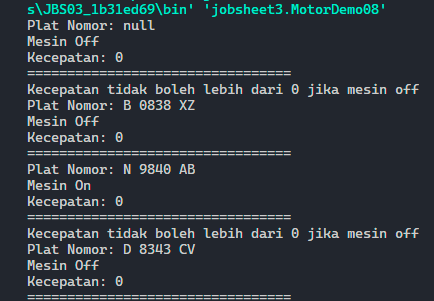




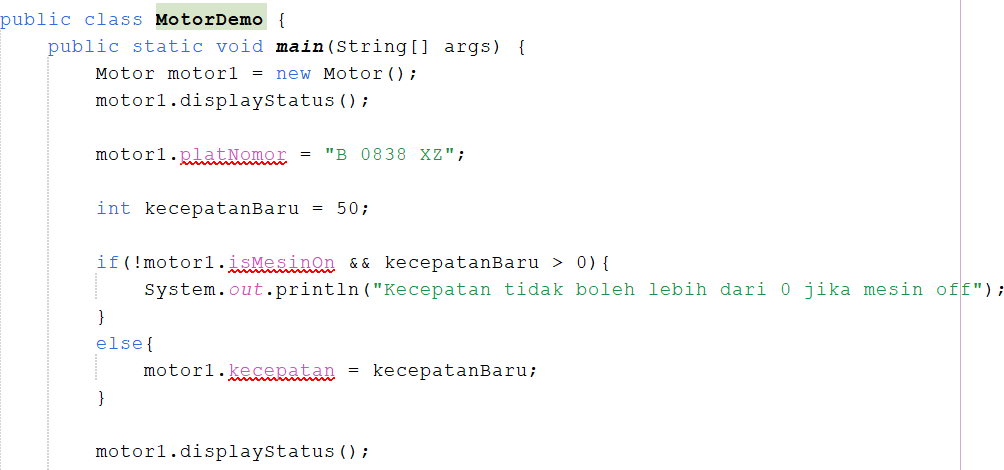


Output:





## Percobaan 2 – Enkapsulasi



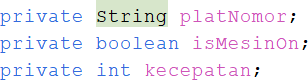
1. Bayangkan bahwa developer baru ingat bahwa seharusnya kecepatan tidak boleh lebih dari 0 jika status mesin tidak menyala setelah membuat 20 objek motor di MotorDemo.java, 10 objek motor di MotorDemo2.java, 25 objek MotorDemo3.java? Pengecekan harus dilakukan 55 kali.
2. Lalu, bagaimana kita bisa memperbaiki class Motor diatas agar dapat digunakan dengan baik? Di sinilah pentingnya melakukan enkapsulasi dalam pemrograman berorientasi objek. Struktur internal class Motor harus disembunyikan dari class lain.

Pada OOP, konsep enkapsulasi diimplementasikan dengan cara:

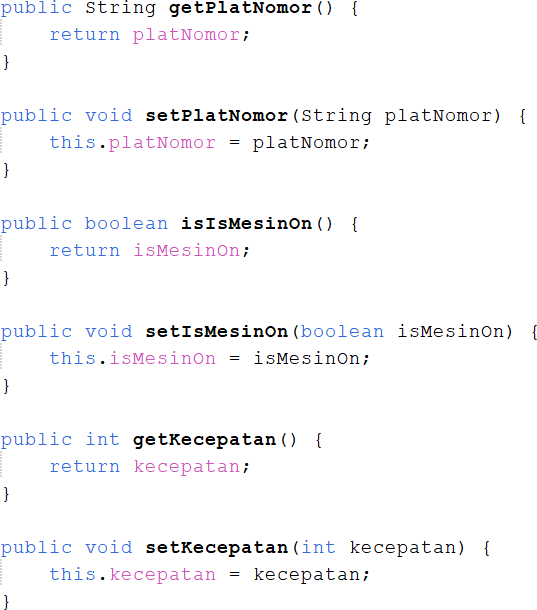
* 1. Menyembunyikan atribut internal (platNomor, isMesinOn, dan kecepatan) dari luar/class lain dengan mengubah access level modifier menjadi private
  2. Menyediakan setter dan getter untuk memanipulasi dan mengakses nilai atribut tersebut

|  |
| --- |
| Motor |
| * platNomor: String * isMesinOn: Boolean * kecepatan: int |
| +displayStatus(): void  +setPlatNomor(platNomor:String): void  +getPlatNomor(): String  +setIsMesinOn(isMesinOn:boolean): void  +getIsMesinOn(): boolean  +setKecepatan(kecepatan:int): void  +getKecepatan(): int |

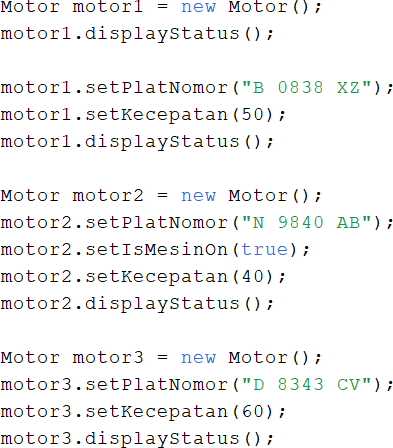
1. Ubah access level modifier menjadi private



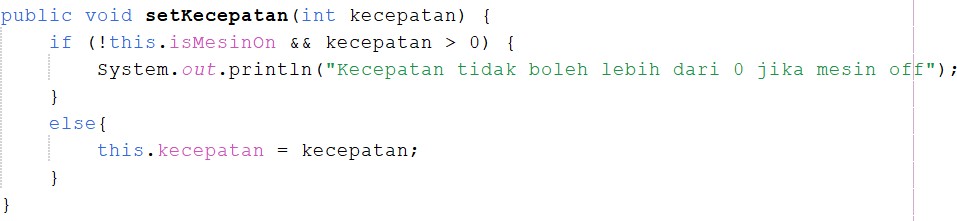
1. Setelah berubah menjadi private, atribut platNomor, isMesinOn, dan kecepatan tidak bisa diakses dari luar class (muncul error)
2. Selanjutnya perlu di buat setter dan getter untuk setiap atribut.



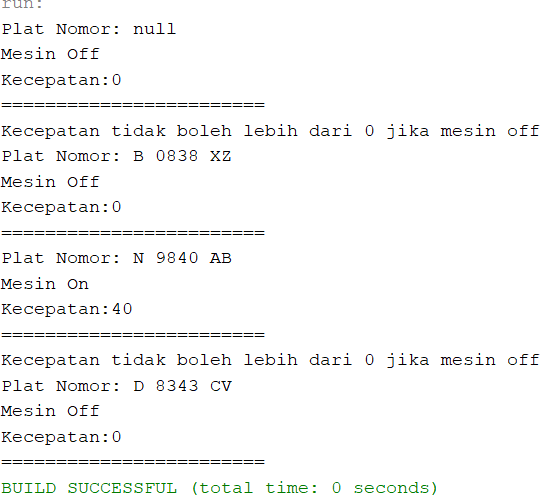
1. Dengan enkapsulasi, nilai atribut diakses menggunakan getter dan dimanipulasi menggunakan setter sebagai berikut (belum ada validasi nilai kecepatan terhadap status mesin)



1. Dengan menerapkan enkapsulasi, perubahan requirement di tengah implementasi program dapat dilakukan dengan lebih mudah. Pada setter kecepatan, dilakukan validasi nilai kecepatan terhadap status mesin sebagai berikut:

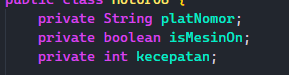


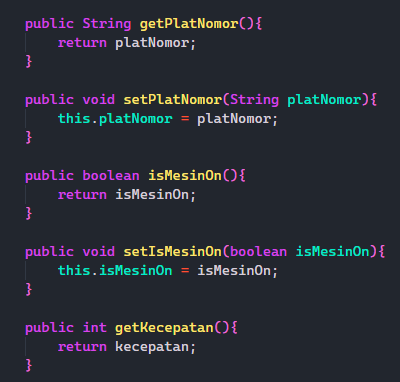
1. Run MotorDemo.java. Hasilnya sebagai berikut:

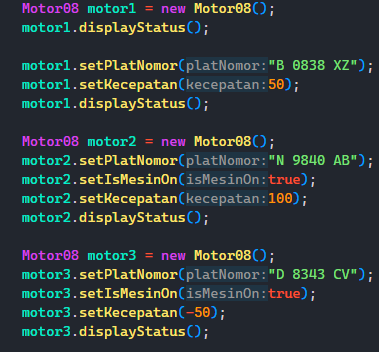


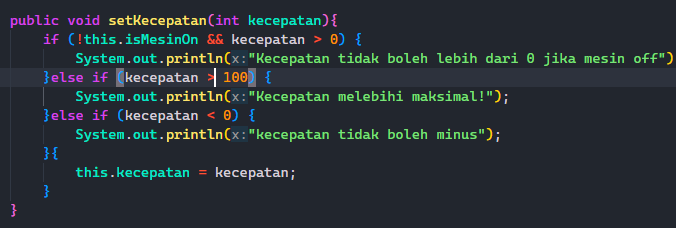
1. Setter dan getter dipakai sebagai “gerbang” untuk mengakses atau memodifikasi atribut yang bernilai private. Hal ini akan membuat kontrol atau validasi atribut lebih mudah dilakukan. Jika ada perubahan requirement di kemudian hari, misalnya atribut kecepatan tidak boleh bernilai negatif, hanya perlu dilakukan modifikasi pada setKecepatan() tanpa perlu melakukan perubahan berulang kali di seluruh program yang melakukan assignment nilai kecepatan motor.

Hasil:

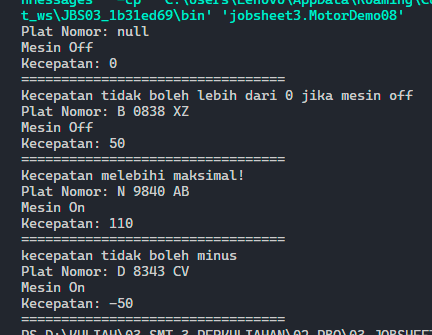








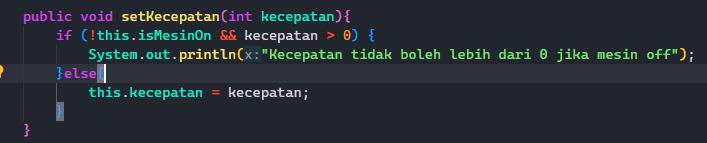
Output:



## Pertanyaan

1. Pada class MotorDemo, saat kita menambah kecepatan untuk pertama kalinya, mengapa muncul peringatan “Kecepatan tidak bisa bertambah karena Mesin Off!”?

**Jawaban:**



Dengan adanya kode diatas kecepatan tidak bisa bertambah apabila kondisi mesin masih mati

1. Mengapat atribut merek, kecepatan, dan statusMesin diset private?

**Jawaban:** agar atribut tersebut tidak bisa digunakan di class lain selain class motor itu sendiri

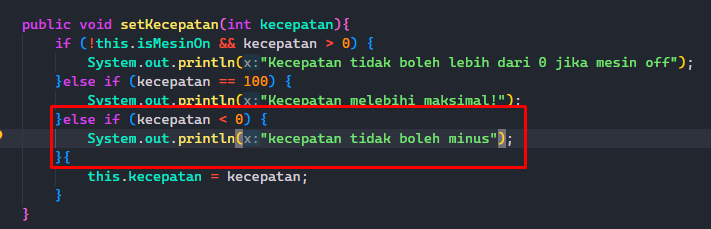
1. Apa fungsi dari setter dan getter?

**Jawaban:** penggunaan setter digunakan untuk memanipulasi nilai dari private atribut yang digunakan sedangkan untuk getter digunakan untuk menampilkan nilai kembalian dari atribut private yang dipilih.

1. Ubah class Motor sehingga kecepatan maksimalnya adalah 100



1. Ubah class Motor sehingga kecepatan nya tidak boleh nilai negatif



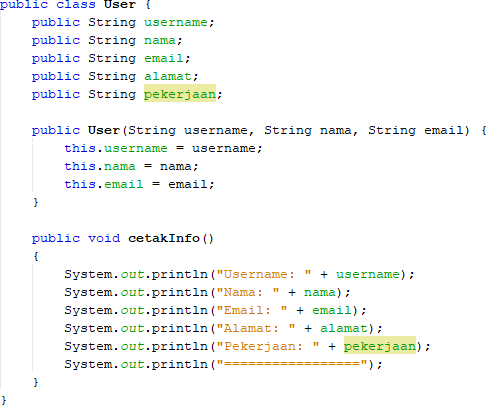
## Percobaan 3 - Constructor

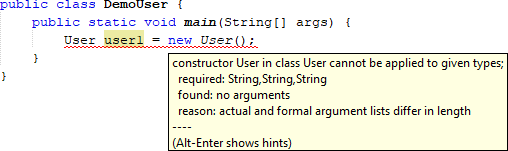
Pada pelajaran sebelumnya, instansiasi objek dari suatu class dilakukan dengan menggunakan syntax **new <NamaClass>();** misalnya motor1 = new Motor();

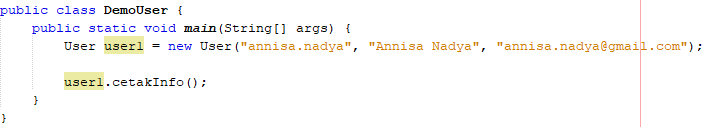
Dengan baris kode tersebut, kita telah menggunakan constructor default yaitu Motor() tanpa parameter apapun. Oleh karena itu, setiap nilai atribut pada motor1 akan bernilai default. Atribut merek yang bertipe string bernilai default **null**, atribut isMesinOn yang bertipe boolean bernilai default **false**, dan atribut kecepatan yang bertipe integer bernilai default **0**.

Pada beberapa kasus, kita menginginkan suatu objek dari class tertentu sudah memiliki nilai untuk beberapa (atau seluruh) atribut pada saat objek tersebut dibuat.

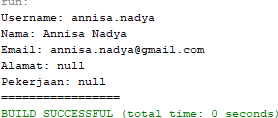
1. Misalkan di sebuah sistem informasi, terdapat class **User** yang memiliki atribut username, nama, email, alamat, dan pekerjaan. Saat suatu objek user dibuat, user tersebut harus sudah memiliki nilai username, nama, dan email. Dengan kebutuhan tersebut, kita harus membuat sebuah constructor baru sebagai berikut:



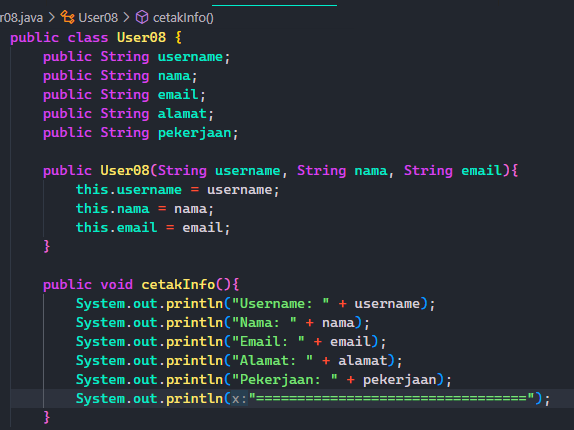
1. Setelah kita menyediakan constructor baru secara eksplisit, maka constructor default yaitu User() tidak bisa digunakan lagi kecuali kita buat juga. Multiple constructor akan dibahas pada materi overloading dan overriding.
2. Instansiasi objek user baru dengan constructor yang telah dibuat pada no 1 bisa dilakukan dengan cara berikut:

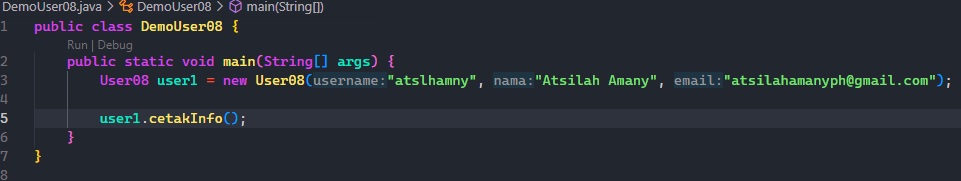


1. Hasilnya sebagai berikut:

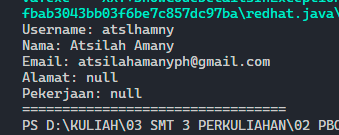


Hasil:





Output:



## Pertanyaan

1. Apa yand dimaksud constructor?

**Jawaban:** merupakan fungsi khusus yang digunakan untuk menginisialiasasi objek saat dibuat **(instansiasi)**

1. Sebutkan aturan dalam membuat constructor?

**Jawaban:** nama method harus sama dengan nama class, hanya bisa dijalankan/dipanggil pada proses instansiasi, bisa memiliki parameter, tidak memiliki tipe data method

1. Lakukan analisa dan buat kesimpulan apakah constructor bisa bertipe private?

**Jawaban: bisa,** konstruktor bisa bertipe private biasanya digunakan untuk mencegah instansiasi objek dari luas class tersebut dan membatasi pembuatan objek hanya didalam class atau dengan control tertentu

# Tugas

1. Pada sebuah sistem informasi koperasi simpan pinjam, terdapat class Anggota yang memiliki atribut antara lain nomor KTP, nama, limit peminjaman, dan jumlah pinjaman. Anggota dapat meminjam uang dengan limit peminjaman yang ditentukan. Anggota juga dapat mengangsur pinjaman. Ketika Anggota tersebut mengangsur pinjaman, maka jumlah pinjaman akan berkurang sesuai dengan nominal yang diangsur.

Buatlah class Anggota tersebut, berikan atribut, method dan constructor sesuai dengan kebutuhan. Uji dengan TestKoperasi berikut ini untuk memeriksa apakah class Anggota yang anda buat telah sesuai dengan yang diharapkan.

*Perhatikan bahwa nilai atribut pinjaman tidak dapat diubah secara random dari luar class, tetapi hanya dapat diubah melalui method pinjam() dan angsur()*

public class TestKoperasi

{

public static void main(String[] args)

{

Anggota anggota1 = new Anggota("111333444", "Donny", 5000000);

System.out.println("Nama Anggota: " + anggota1.getNama()); System.out.println("Limit Pinjaman: " + anggota1.getLimitPinjaman());

System.out.println("\nMeminjam uang 10.000.000..."); anggota1.pinjam(10000000);

System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

System.out.println("\nMeminjam uang 4.000.000..."); anggota1.pinjam(4000000);

System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

System.out.println("\nMembayar angsuran 1.000.000"); anggota1.angsur(1000000);

System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

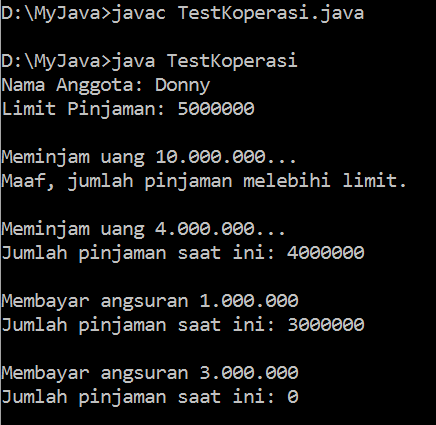
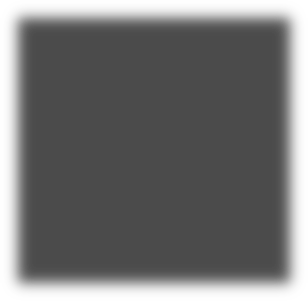
System.out.println("\nMembayar angsuran 3.000.000"); anggota1.angsur(3000000);

System.out.println("Jumlah pinjaman saat ini: " + anggota1.getJumlahPinjaman());

}

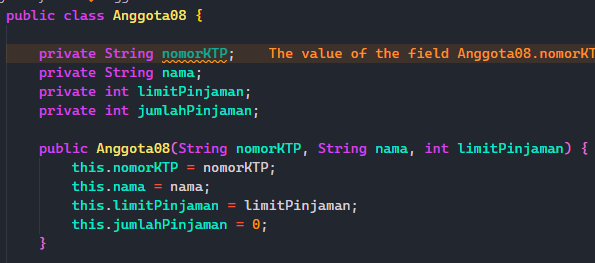
}

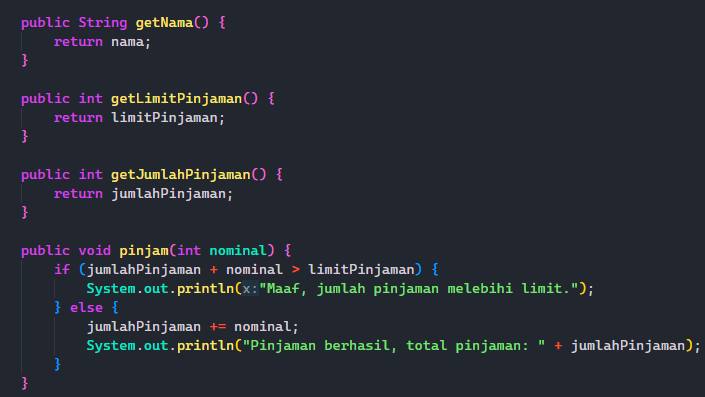
Hasil yang diharapkan:

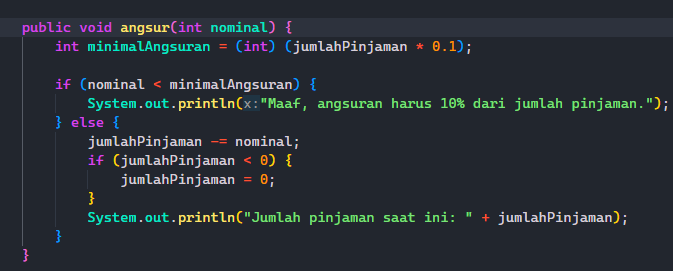


1. Modifikasi class Anggota agar nominal yang dapat diangsur minimal adalah 10% dari jumlah pinjaman saat ini. Jika mengangsur kurang dari itu, maka muncul peringatan “Maaf, angsuran harus 10% dari jumlah pinjaman”.

Hasil:









Output:

