Noordin Asnawi, M.Kom

TEORI DAN PRAKTIK OBJECT ORIENTED PROGRAMMING MENGGUNAKAN JAVA

UNIPMA Press
WE GOT IT

TEORI DAN PRAKTIK

OBJECT ORIENTED PROGRAMMING

MENGGUNAKAN **JAVA**

Noordin Asnawi, M.Kom

Buku ini membahas mengenai pengenalan dasar dari pemrograman berorientasi objek (PBO) yang menggunakan bahasa pemrograman Java dalam praktiknya, mulai dari teori-teori mengenai PBO, teori dari bahasa pemrograman Java, sampai dengan praktik atau implementasi PBO dengan bahasa Java.

Dalam buku ini terdapat langkah-langkah sebagai pembelajaran dalam pengenalan dasar dari PBO dan bahasa pemrograman Java seperti berikut:

1.Dasar PBO
2.Pengenalan Bahasa Pemrograman Java
3.Persiapan Penggunaan Java
4.Struktur Percabangan (Decision)
5.Struktur Perulangan (Looping)
6.Struktur Array
7.Method dan Parameter
8.Implementasi PBO
9.Penggunaan Basisdata (Database)

Semoga buku ini dapat memberikan manfaat bagi pembacanya, khususnya mahasiswa yang mempelajari mengenai PBO dengan bahasa pemrograman Java. Penulis berharap kritik dan saran yang membangun untuk memperbaiki buku ini supaya lebih baik lagi..

UNIPMA Press
WE GOT IT

UNIPMA Press
Universitas PGRI Madiun
Jl. Setia Budi no. 85 Madiun, Jawa Timur 63118
Telp. (0351) 462386, Fax. (0351) 459400
E-mail: upress@unipma.ac.id
Website: kwu.unipma.ac.id

ISBN 978-623-8095-33-9
9 786238 095339

TEORI DAN PRAKTIK OBJECT ORIENTED PROGRAMMING

MENGGUNAKAN JAVA

Noordin Asnawi, M.Kom.

UNIPMA Press
WE GOT IT

TEORI DAN PRAKTIK OBJECT ORIENTED PROGRAMMING MENGGUNAKAN JAVA

Penulis:
Noordin Asnawi, M.Kom.

Editor:
Ridho Pamungkas, M.Kom.

Desain Sampul:
Rizki Bindra Permana, S.Kom

Penata Letak:
Mei Lenawati, M.Kom.

Cetakan Pertama, November 2023

Diterbitkan Oleh:
UNIPMA Press Universitas PGRI Madiun
JI. Setiabudi No. 85 Madiun Jawa Timur 63118
Telp. (0351) 462986, Fax. (0351) 459400
E-Mail: upress@unipma.ac.id
Website: www.kwu.unipma.ac.id

ISBN: 978-623-8095-33-9

KATA PENGANTAR

Puji syukur penulis haturkan kepada Allah SWT atas limpahan rahmat-Nya dan ridho-Nya, sehingga penulis bisa menyelesaikan Buku Ajar dengan judul "Teori dan Praktik Object Oriented Programming Menggunakan Java". Tujuan disusunnya buku ajar ini adalah supaya mahasiswa dapat mengetahui dasar-dasar dari pemrograman berorientasi objek dengan menggunakan Java yang merupakan salah satu bahasa pemrograman yang terkenal diantara banyak macam bahasa pemrograman. Dimana pemrograman Java merupakan bahasa pemrograman yang mempunyai kelebihan-kelebihan tertentu yang membuat bahasa pemrograman ini dipelajari dan digunakan oleh banyak orang.

Tersusunnya buku ajar ini tentu bukan dari usaha penulis sendiri, dukungan moral dan material dari berbagai pihak sangatlah membantu tersusunnya buku ajar ini. Untuk itu penulis ucapkan terima kasih yang sebesar-besarnya kepada keluarga, rekan-rekan, dan pihak-pihak lainnya yang turut membantu secara moral dan material bagi tersusunnya buku ajar ini.

Buku ajar yang penulis susun selama ini masih sangatlah jauh dari kata sempurna, untuk itu penulis harapkan kritik dan saran yang membangun guna memperbaiki buku ajar ini untuk kedepannya bisa lebih baik lagi dan lebih bermanfaat.

Madiun, 27 September 2023

Penulis

DAFTAR ISI

BAB 1
DASAR PEMROGRAMAN BERORIENTASI OBJEK

1.1 Pendahuluan

Pengembangan perangkat lunak adalah penerapan berbagai *tools*, teknik, dan metodologi pemrograman untuk desain dan pengembangan perangkat lunak. Motivasi utama untuk pengembangan berkelanjutan dari *tools* ini adalah untuk menangani peningkatan kompleksitas sistem perangkat lunak. Tujuan utama pengembang adalah untuk mengembangkan program yang kompleks dengan mudah dan efisien.

Sistem yang kompleks bersifat hierarkis dengan beberapa level dan setiap level mewakili level yang berbeda dengan batas yang jelas antar level. Meningkatnya kompleksitas sistem perangkat lunak disebabkan oleh peningkatan kebutuhan pengguna. Kompleksitas dalam sistem perangkat lunak bersifat arbitrer (manasuka) yang bervariasi dari aplikasi ke aplikasi.

1.2 Bahasa Pemrograman

Komputer tidak mengerti bahasa manusia, jadi program harus ditulis dalam sebuah bahasa komputer supaya dapat digunakan. Ada ratusan bahasa pemrograman, dan mereka dikembangkan untuk membuat proses

pemrograman lebih mudah bagi orang-orang. Namun, semua program harus dikonversi ke dalam bahasa yang dapat dimengerti oleh komputer. Pemrograman komputer adalah seni menulis kode yang dapat dieksekusi komputer untuk menyelesaikan tugas tertentu. Pemrograman komputer memungkinkan kita untuk mengendalikan perangkat komputer. Kita memberi tahu komputer apa yang harus dilakukan dengan menulis dan menginstal program yang dapat dieksekusi yang dijalankan oleh komputer berdasarkan permintaan. Pemrograman komputer telah berkembang sejak awal, alasan dari evolusi ini, adalah kebutuhan untuk membuat bahasa pemrograman lebih ramah pengguna.

Pemrograman komputer telah melalui berbagai tahap perkembangan, tahap pertama dikenal sebagai bahasa generasi pertama, ini adalah instruksi data dasar (bahasa mesin). Kode ditulis langsung untuk diproses oleh komputer; artinya tidak ada kompilasi kode sebelum eksekusi. Bahasa generasi kedua seperti bahasa assembly menggunakan assembler untuk mengubah pernyataan bahasa menjadi bahasa mesin. Bahasa generasi ketiga juga dikenal sebagai bahasa tingkat tinggi menggunakan kompiler untuk mengubah bahasa tingkat tinggi. Beberapa contohnya adalah C, C++, Java, dll. Bahasa generasi keempat sangat mirip dengan bahasa manusia.

Contohnya adalah bahasa pemrograman SQL. Bahasa generasi kelima menggunakan antarmuka grafis untuk membuat pernyataan bahasa.

1.2.1 Bahasa Mesin

Bahasa asli komputer, yang berbeda di antara berbagai jenis komputer, adalah bahasa mesin—satu set instruksi primitif bawaan. Instruksi ini dalam bentuk kode biner, jadi jika ingin memberikan komputer instruksi dalam bahasa aslinya, maka harus memasukkan instruksi sebagai kode biner. Misalnya, untuk menambahkan dua angka, menulis instruksi dalam kode biner, seperti ini:

1101101010011010

1.2.2 Bahasa Assembly

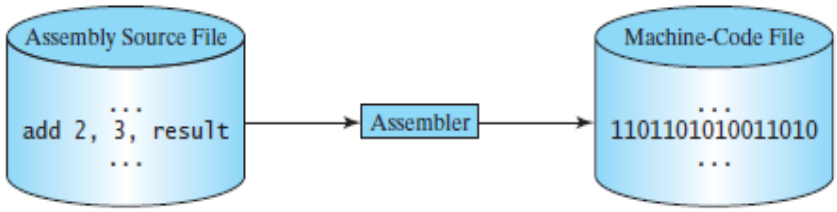
Penyempurnaan pertama diusulkan dalam bentuk bahasa rakitan. Menggunakan bahasa assembly, seorang programmer dapat 'mewakili' komputasi menggunakan mnemonik tekstual untuk mewakili instruksi dan data (yaitu, lokasi memori). Perangkat lunak khusus yang disebut assembler dapat menerjemahkan representasi seperti itu, yaitu program bahasa rakitan, ke dalam satu set instruksi mesin. Meskipun keuntungan untuk dapat merepresentasikan program menggunakan mnemonik tekstual, program bahasa rakitan untuk masalah yang cukup kompleks sulit didapat

dari domain, dan sama sulitnya untuk dipahami. Program semacam itu tetap 'bergantung pada mesin'—program yang ditulis untuk model tertentu dari komputer merek tertentu mungkin tidak portabel ke mesin lain dari merek atau model yang berbeda.

Pemrograman dalam bahasa mesin adalah proses yang membosankan. Selain itu, program yang ditulis dalam bahasa mesin sangat sulit untuk dibaca dan dimodifikasi. Untuk alasan ini, bahasa rakitan diciptakan pada hari-hari awal komputasi sebagai alternatif untuk bahasa mesin. Perakitan bahasa menggunakan kata deskriptif pendek, yang dikenal sebagai mnemonik, untuk mewakili masing-masing instruksi bahasa mesin. Misalnya, *mnemonic add* biasanya berarti menambahkan angka dan sub berarti mengurangi angka. Untuk menjumlahkan angka 2 dan 3 dan mendapatkan hasilnya, instruksi ditulis dalam kode assembly seperti ini:

```
add 2, 3, result
```

Bahasa assembly dikembangkan untuk membuat pemrograman lebih mudah. Namun, karena komputer tidak dapat memahami bahasa assembly, program lain—disebut assembler—adalah digunakan untuk menerjemahkan program bahasa rakitan ke dalam kode mesin, seperti yang ditunjukkan pada Gambar 1.1.



Gambar 1.1 Assembler menerjemahkan instruksi bahasa assembly ke dalam kode mesin

Menulis kode dalam bahasa assembly lebih mudah daripada dalam bahasa mesin. Namun, itu masih membosankan untuk menulis kode dalam bahasa assembly. Sebuah instruksi dalam bahasa assembly pada dasarnya sesuai dengan instruksi dalam kode mesin. Menulis dalam perakitan mengharuskan untuk tahu cara kerja CPU. Bahasa assembly disebut sebagai bahasa tingkat rendah, karena bahasa assembly mirip dengan bahasa mesin dan bergantung pada mesin.

1.2.3 Bahasa Tingkat Tinggi

Pada tahun 1950-an, generasi baru bahasa pemrograman yang dikenal sebagai bahasa tingkat tinggi muncul. Mereka adalah platform-independen, yang berarti dapat menulis program di bahasa tingkat tinggi dan menjalankannya di berbagai jenis mesin. Bahasa tingkat tinggi seperti bahasa Inggris dan mudah dipelajari dan digunakan. Instruksi dalam bahasa pemrograman tingkat tinggi disebut pernyataan. Di sini, misalnya, adalah

pernyataan bahasa tingkat tinggi yang menghitung luas sebuah lingkaran dengan jari-jari 5:

```
Luas_lingkaran = 3.1415 * 5 * 5
```

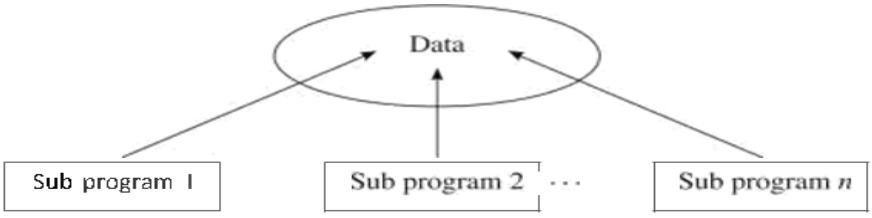
1.3 Perkembangan Bahasa Tingkat Tinggi

1. Bahasa Generasi Pertama

Bahasa-bahasa ini terutama digunakan untuk aplikasi ilmiah dan teknik. Bahasa ini menyediakan fitur untuk perhitungan ilmiah dan domain aplikasi mereka dibatasi hanya untuk perhitungan matematis. Bahasa-bahasa ini lebih unggul daripada bahasa mesin dan bahasa rakitan yang bergantung pada mesin. Dalam bahasa ini, program tidak lain adalah serangkaian langkah dengan setiap langkah melakukan beberapa operasi makna. Beberapa bahasa generasi ini adalah FORTRAN I, ALGOL, IPL V, dll. Bahasa-bahasa ini cocok untuk masalah kecil, tetapi karena ukuran masalah meningkat, mereka gagal menghasilkan hasil yang diinginkan. Dibandingkan dengan mesin dan bahasa rakitan, mereka lebih berorientasi pada masalah daripada mesin.

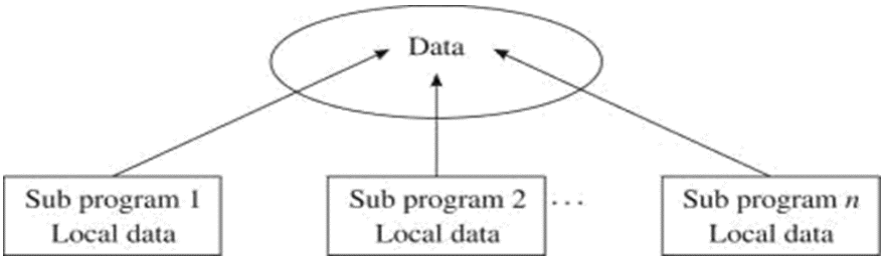
2. Bahasa Generasi Kedua

Salah satu kelemahan utama bahasa generasi pertama adalah kemampuannya yang terbatas dan area aplikasi yang terbatas. Pada saat ini, kemampuan komputer meningkat dan mesin jauh lebih kuat daripada generasi sebelumnya dan karenanya domain masalah diperluas dan komputer digunakan untuk aplikasi bisnis juga. Bahasa yang dikembangkan selama periode ini lebih menekankan pada abstraksi fungsi dan karenanya lebih cocok untuk memecahkan masalah yang lebih besar. Di sini fokus utamanya adalah memberi tahu mesin apa yang harus dilakukan seperti membaca dari file, mengurutkan konten dan menyalin *output* ke file, dll. Struktur umum bahasa generasi pertama dan awal kedua seperti yang ditunjukkan pada Gambar 1.1 di mana blok bangunan adalah sub program dan datanya bersifat global. Data tersebut dapat diakses oleh setiap sub program. Karena ketergantungan sub program pada data global, kesalahan di satu bagian program memiliki efek konsekuensial pada bagian lain yang membuat modifikasi program menjadi sulit. Selain itu, integritas desain asli akan hilang pada saat modifikasi dilakukan pada sub program.



Gambar 1.1 Topologi bahasa generasi pertama dan awal generasi kedua

Selama akhir generasi kedua, bahasa yang mendukung perpindahan parameter ditemukan dan konsep struktur kontrol dan konsep ruang lingkup deklarasi variabel berkembang yang memunculkan jenis abstraksi lain yang dikenal sebagai abstraksi prosedur yang memungkinkan pemrogram untuk menulis program yang jauh lebih kompleks. Topologi bahasa akhir generasi kedua dan awal generasi ketiga seperti yang ditunjukkan pada Gambar 1.2.



Gambar 1.2 Struktur umum bahasa akhir generasi kedua dan awal generasi ketiga

3. Bahasa Generasi Ketiga

Dalam periode ini, biaya perangkat keras komputer berkurang secara drastis tetapi kemampuan komputasi telah meningkat secara eksponensial dan karenanya domain masalah komputer melebar. Bahasa pemrograman

generasi ini mampu menangani masalah yang jauh lebih besar. Bahasa ini juga dikenal sebagai bahasa berorientasi prosedur. Contoh: C, Pascal, dll.

Di sini masalah yang diberikan didekomposisi menjadi jumlah subtugas. Masing-masing subtugas ini dapat diselesaikan dengan bantuan fungsi yang ditulis untuk tujuan tersebut. Sebuah program dalam bahasa berorientasi prosedur tidak lain adalah seperangkat fungsi. Setiap fungsi memiliki tujuan yang terdefinisi dengan baik dan antarmuka yang terdefinisi dengan baik untuk fungsi lainnya. Suatu fungsi dapat memiliki datanya sendiri yang dikenal sebagai data lokal dan beberapa fungsi dapat mengakses data yang sama yang dikenal sebagai data global. Pendekatan membagi program besar menjadi beberapa fungsi yang lebih kecil dan lebih sederhana dikenal sebagai pemrograman terstruktur. Pendekatan pemrograman terstruktur mampu menangani kompleksitas sampai batas tertentu yang memungkinkan membangun bagian yang berbeda dari program yang sama secara mandiri. Modul program yang berbeda dapat dikompilasi secara terpisah.

Teknik pemrograman terstruktur banyak digunakan teknik pemrograman sampai munculnya pendekatan berorientasi objek. Kemampuan pemrograman terstruktur untuk menangani program kompleks yang besar

menurun seiring dengan bertambahnya ukuran dan kompleksitas program melebihi batas tertentu. Beberapa kelemahan utama dari bahasa berorientasi prosedur adalah:

a. Karena setiap fungsi memiliki akses ke data global, sulit untuk melacak perubahan data global dalam program besar. Data global tidak sepenuhnya diamankan dalam bahasa berorientasi prosedur.
b. Setiap kali data global diubah, perlu untuk memodifikasi semua fungsi yang mengakses data itu dan karenanya modifikasi program adalah mimpi buruk programmer.
c. Mereka tidak memodelkan masalah dunia nyata secara efisien. Tidak ada korespondensi satu-satu antara elemen masalah dan fungsi. Tidak setiap elemen masalah dapat ditulis sebagai fungsi karena fungsi berorientasi pada tindakan.
d. Dukungan terbatas untuk abstraksi data dan deteksi kesalahan hanya dimungkinkan selama eksekusi yang tidak diinginkan untuk mengembangkan program yang kompleks.

Beberapa karakteristik bahasa berorientasi prosedur adalah:

a. Pentingnya algoritma daripada data.

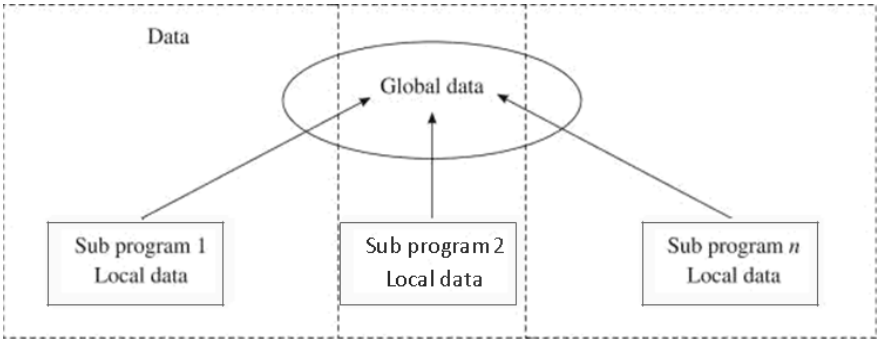
b. Program yang kompleks dibagi menjadi beberapa *function* yang lebih kecil dan lebih sederhana untuk meminimalkan kompleksitas.

c. *Function* dapat memiliki data lokalnya sendiri dan juga memiliki akses ke data global.

d. Mengadopsi pendekatan *top-down*.

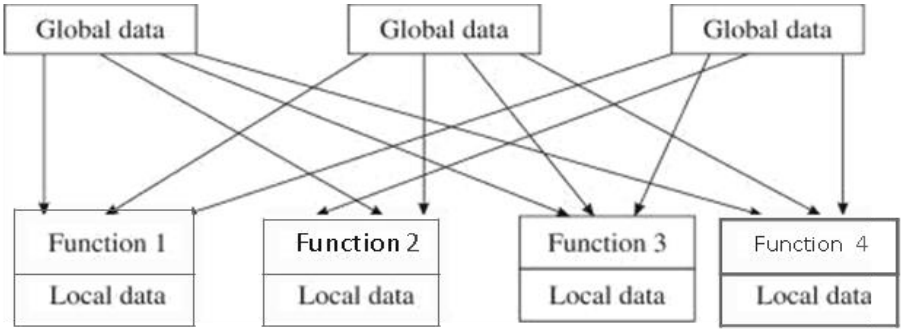
e. *Function* dapat berkomunikasi satu sama lain melalui *argument passing.*

Topologi bahasa akhir generasi ketiga ditunjukkan pada Gambar 1.3.



Gambar 1.3 Struktur umum bahasa akhir generasi ketiga

Hubungan antar *function* dalam bahasa berorientasi prosedur ditunjukkan pada Gambar 1.4.



Gambar 1.4 Hubungan antar *function* dalam bahasa berorientasi prosedur

4. Bahasa Generasi Keempat

Bahasa generasi keempat berurusan dengan dua bidang berikut yang menjadi semakin penting yaitu database dan bahasa kueri, dan generator program atau aplikasi. Tingkat yang sangat tinggi atau bahasa berorientasi masalah, juga disebut bahasa generasi keempat (4 GLs), jauh lebih berorientasi pengguna dan memungkinkan pengguna untuk mengembangkan program dengan perintah yang lebih sedikit dibandingkan dengan bahasa prosedural, meskipun mereka membutuhkan lebih banyak daya komputasi. Bahasa ini dikenal sebagai bahasa yang berorientasi pada masalah karena dirancang untuk memecahkan masalah tertentu, sedangkan bahasa prosedural adalah bahasa tujuan yang lebih umum.

Salah satu bahasa kueri basis data yang paling terkenal adalah SQL (*Structure Query Language*): bahasa kueri untuk basis data relasional yang

dikembangkan IBM dan yang didasarkan pada persyaratan kode untuk bahasa kueri non-prosedural untuk basis data relasional. NATURAL adalah pendekatan lain dalam bidang ini yang menekankan pada gaya pemrograman terstruktur. Program atau aplikasi generator sering didasarkan pada metode spesifikasi tertentu dan menghasilkan output (misalnya program tingkat tinggi) dengan spesifikasi yang sesuai.

Tiga jenis bahasa berorientasi masalah adalah *report generators*, *query languages*, dan *application generators*.

a. ***Report Generators***. Pembuat laporan, juga dikenal sebagai penulis laporan, adalah program untuk pengguna akhir yang menghasilkan laporan. Laporan dapat berupa cetakan atau tampilan layar. Ini mungkin menunjukkan semua atau sebagian dari file database. Kita dapat menentukan format di muka-kolom, judul, dll dan pembuat laporan kemudian akan menghasilkan data dalam format itu.

b. ***Query Languages***. Bahasa query adalah bahasa yang mudah digunakan untuk mengambil data dari sistem manajemen *database*. Query dapat diberikan dalam bentuk kalimat atau perintah mendekati bahasa Inggris. Atau query dapat diperoleh dari pilihan pada menu.

Application generators. Generator aplikasi adalah alat pemrogram yang terdiri dari modul yang telah diprogram sebelumnya untuk menyelesaikan berbagai tugas. Manfaatnya adalah programmer dapat menghasilkan program aplikasi dari deskripsi masalah daripada dengan pemrograman tradisional, di mana ia harus menentukan bagaimana data harus diproses.

5. Bahasa Generasi Kelima

Bahasa alami ada dua jenis. Yang pertama adalah bahasa manusia biasa: Inggris, Indonesia, dll. Yang kedua adalah bahasa pemrograman yang menggunakan bahasa manusia untuk menyediakan orang atau koneksi yang lebih alami dengan komputer. Bahasa alami adalah bagian dari bidang studi yang dikenal sebagai kecerdasan buatan. Kecerdasan buatan (*Artificial Inteligence*) adalah sekelompok teknologi terkait yang mencoba mengembangkan mesin yang mampu meniru kualitas seperti manusia, seperti belajar, menalar, berkomunikasi, melihat, dan mendengar.

1.3.1 Topologi Umum Bahasa Berbasis Objek dan Berorientasi Objek

Kelemahan utama dari semua generasi sebelumnya adalah mereka hanya mendukung abstraksi fungsi. Abstraksi data memainkan peran penting

dalam menguasai kompleksitas. Banyak aplikasi yang kompleks karena memerlukan manipulasi beberapa objek data yang mengharuskan program untuk mencari berbagai jenis bahasa yang mendukung abstraksi data yang telah memunculkan evolusi bahasa berorientasi objek dan berbasis objek, misalnya, C++, Smalltalk, Java, dll.

1.4 Paradigma Pemrograman Berorientasi Objek

Pendekatan berorientasi objek adalah solusi utama untuk kekurangan yang dihadapi programmer dalam pendekatan berorientasi prosedur. Gagasan utama di balik bahasa pemrograman berorientasi objek adalah untuk merangkum data dan fungsi yang beroperasi pada data ini ke dalam unit bermakna yang dikenal sebagai objek. Dalam Pemrograman Berorientasi Objek (OOP), data dianggap sebagai elemen penting dan terikat dengan fungsi yang beroperasi di atasnya.

Kumpulan data yang dienkapsulasi dalam suatu objek dikenal sebagai data anggota dan fungsi yang beroperasi pada data ini dikenal sebagai fungsi anggota. Di OOP, data tidak dapat bergerak bebas di sekitar sistem dan data hanya dapat diakses dan dimodifikasi oleh fungsi anggota. Fungsi eksternal tidak dapat mengakses data anggota suatu objek dan karenanya data

dilindungi dari modifikasi oleh fungsi eksternal. OOP memperlakukan data sebagai entitas penting dibandingkan dengan pendekatan berorientasi prosedur di mana penekanannya adalah pada fungsi daripada data.

Data disembunyikan dalam pemrograman berorientasi objek dan karenanya aman dari modifikasi yang tidak disengaja. Anggota data dari suatu objek tertentu hanya dapat diakses oleh fungsi anggota dari objek tertentu sedangkan fungsi anggota dari suatu objek dapat diakses oleh fungsi anggota dari objek lain.

Pemrograman berorientasi objek adalah pendekatan baru untuk pemrograman melalui modularisasi program dengan membuat area terpisah untuk data dan fungsi yang digunakan untuk membuat salinan modul tersebut sesuai permintaan. Beberapa perbedaan utama antara pemrograman berorientasi prosedur dan pemrograman berorientasi objek diberikan pada Tabel 1.1.

Tabel 1.1 Perbedaan antara pendekatan berorientasi prosedur dan pendekatan berorientasi objek

Pendekatan Berorientasi Prosedur	**Pendekatan berbasis Objek**
Program didekomposisi menjadi beberapa fungsi	Program didekomposisi menjadi beberapa unit yang bermakna dikenal sebagai objek

Fungsi diberi prioritas lebih tinggi daripada data	Data diberi prioritas lebih tinggi daripada fungsi
Mengadopsi pendekatan desain *top down*	Mengadopsi pendekatan desain *bottom up*
Fungsi berkomunikasi dengan setiap pesanan melalui argumen	Objek berkomunikasi melalui fungsi anggota
Fungsi adalah kumpulan instruksi yang melakukan tugas tertentu	Objek adalah kumpulan data dan fungsi
Fungsi tidak memodelkan objek dunia nyata secara efisien	Objek memodelkan objek dunia nyata secara efisien
Data dapat berpindah dengan mudah dari satu fungsi ke fungsi lainnya	Data tidak dapat keluar dari objek

1.4.1 Prinsip Dasar Pemrograman Berorientasi Objek

Beberapa karakteristik pemrograman berorientasi objek adalah:

1. **Objek:** Objek adalah entitas run time dasar dalam pemrograman berorientasi objek. Objek dunia nyata apa pun dapat diperlakukan sebagai objek di OOP. Dalam sistem perguruan tinggi, misalnya mahasiswa, mata kuliah dan fakultas dapat dijadikan objek. Demikian pula dalam struktur data, Stack, Queue, Tree bisa menjadi objek. Dalam rekening bank, nasabah bisa menjadi objeknya. Objek yang dipilih harus memiliki kecocokan yang dekat dengan objek dunia nyata. Pemetaan yang dekat antara objek dunia nyata dan konstruksi pemrograman tidak ada dalam pemrograman berorientasi prosedur. Dalam OOP,

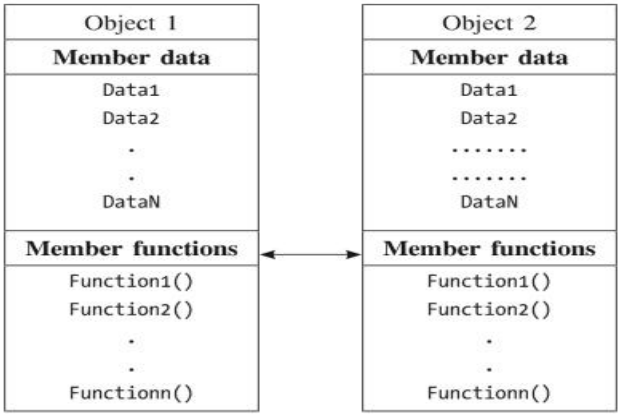
sangat penting untuk menentukan data anggota dan fungsi anggota dari suatu objek mereka dalam pemrograman. Masalah yang diberikan dianalisis dalam hal objek dan bagaimana mereka berkomunikasi dengan objek lain. Setiap objek harus menjadi turunan dari kelas tertentu. Misalnya, mobil dan bus adalah objek kelas kendaraan. Objek mengambil ruang di memori dan itu disebut dalam program dengan namanya. Tipe data bahasa pemrograman sejalan dengan kelas dan variabel tipe data tertentu sejalan dengan objek.

Contoh: Perhatikan pernyataan deklarasi berikut dalam bahasa C.

```
int x, y, z;
float c, d;
```

Di sini variabel x, y dan z semuanya termasuk dalam kategori yang sama int dan masing-masing dari mereka membutuhkan 2 byte memori dalam mesin 16-bit dan demikian pula variabel c dan d milik tipe float dan masing-masing mengambil 4 byte memori. Di sini int dan float setara dengan kelas dan x, y, z. c dan d mirip dengan objek.

Topologi bahasa berorientasi objek dengan hubungan antara data dan fungsi dalam sistem berorientasi objek biasanya terlihat seperti pada Gambar 1.7.



Gambar 1.7 Hubungan antara data dan fungsi

2. **Kelas:** Kelas adalah cara mengelompokkan objek yang memiliki karakteristik serupa. Kelas menyediakan template atau rencana untuk membuat objek dengan tipe tertentu. Untuk membuat objek dari kelas tertentu, kita perlu menentukan data dan fungsi anggota dari kelas tertentu. Misalnya mobil dan sepeda motor adalah objek kelas kendaraan karena semua objek ini memiliki karakteristik kendaraan. Demikian pula, burung beo dan penguin adalah objek kelas burung. Mendefinisikan kelas tidak akan membuat objek apa pun dan hanya objek yang membutuhkan ruang di memori.

3. **Enkapsulasi:** Salah satu fitur mencolok dari program berorientasi objek adalah bahwa data disembunyikan dan tidak dapat diakses oleh fungsi eksternal apa pun. Pembungkusan data dan fungsi ke dalam struktur data yang bermakna yang disebut kelas dikenal sebagai enkapsulasi data. Enkapsulasi adalah cara untuk mencapai keamanan data dalam program berorientasi objek. Melalui enkapsulasi, dimungkinkan untuk mencegah data dari modifikasi yang tidak disengaja oleh fungsi non-anggota eksternal dari suatu kelas. Hanya fungsi anggota yang didefinisikan di dalam kelas yang akan memiliki akses ke data.

4. ***Inheritance*** **(Pewarisan):** Dalam OOP, kelas baru dapat dibangun dari kelas yang ada tanpa mempengaruhi kelas yang ada. Kelas baru akan memiliki fitur dari kelas tingkat yang lebih tinggi dari mana ia diturunkan dan selain itu dapat memiliki fitur sendiri. Kemampuan objek dari satu kelas untuk memperoleh karakteristik objek dari kelas lain dikenal sebagai pewarisan. Kelas yang ada dikenal sebagai kelas dasar dan kelas baru yang diturunkan dari kelas yang ada dikenal sebagai kelas turunan. Proses menurunkan satu kelas dari kelas lain dapat diperluas ke level mana pun. Misalnya, kelas rekening tabungan

dapat diturunkan dari kelas rekening. Melalui pewarisan, dimungkinkan untuk memanfaatkan detail kelas yang ada tanpa mengulangi detail kelas dasar lagi di kelas turunan dan karenanya memberikan keuntungan dapat digunakan kembali dan menghemat waktu pengembangan. Programmer dapat menambahkan fitur baru ke kelas turunan tanpa mempengaruhi kelas dasar.

5. **Polimorfisme:** Polimorfisme adalah fitur penting lain dari pemrograman berorientasi objek di mana suatu fungsi dapat mengambil beberapa bentuk berdasarkan jenis argumen, jumlah argumen dan operator dapat mengambil perilaku yang berbeda tergantung pada operan di mana operator yang sesuai diterapkan. Kemampuan operator dan fungsi untuk mengambil beberapa bentuk ini dikenal sebagai polimorfisme. Ketika operator berperilaku berbeda berdasarkan operan, kami mengatakan bahwa operator kelebihan beban. Demikian pula, fungsi dikatakan kelebihan beban ketika nama fungsi yang sama digunakan untuk banyak tugas dalam program yang sama dengan memvariasikan jenis dan nomor argumen. Objek dengan

struktur internal yang bervariasi dapat berbagi antarmuka eksternal yang sama.

BAB 2
PENGENALAN BAHASA PEMROGRAMAN JAVA

2.1 Sejarah Java

Bahasa pemrograman Java adalah bahasa pemrograman berorientasi objek (Program yang diatur meliputi kelas, data, dan objek) yang dibuat oleh sekelompok kecil insinyur Sun Microsystem yang dikenal sebagai *green tree* di awal tahun sembilan puluhan. Tim ini dipimpin oleh James Gosling, Mike Sheridan, dan Patrick Naughton. Java pertama kali diperkenalkan pada tahun 1995 sebagai bahasa pemrograman berorientasi objek yang sederhana dan aman. Ini adalah bahasa yang unik, karena sebagai bahasa baru pada saat itu, mampu menarik banyak minat dari komunitas komputasi. Dalam dua tahun setelah Java diluncurkan, diperkirakan ada 400.000 programmer Java dan lebih dari 100 buku tentang pemrograman Java.

Gambar 2.1 James Gosling

Melihat pematangan teknologi Web, dan kemampuan multiplatform Java, yang memungkinkan program Java untuk dieksekusi di komputer mana pun, sangat menarik, terutama di jaringan terbuka seperti Internet. Java diimplementasikan melalui kompilasi bagian dan eksekusi selanjutnya pada penerjemah yang diimplementasikan dalam perangkat lunak. Oleh karena itu, aplikasi Java adalah kode objek portabel selama mesin virtual Java diimplementasikan untuk mesin target.

Java adalah bahasa berorientasi objek, tidak seperti Pasal dan C, yang merupakan bahasa prosedural. Sebagai seorang programmer, pemrograman berorientasi objek berarti kita berfokus pada pembuatan kelas untuk mewakili data dalam aplikasi, bukan pada solusi untuk masalah sebagai serangkaian prosedur yang harus diikuti dalam urutan yang

ditetapkan. Sintaks bahasa Java banyak meniru sintaks yang terdapat pada bahasa C dan C++ hanya saja model objek dibuat lebih sederhana, hal ini dikarenakan bahasa Java merupakan pengembangan dari bahasa C dan C++. Java sejak awal sudah dirancang sebagai bahasa pemrograman yang bisa berjalan diberbagai komputer termasuk telepon genggam.

Versi awal Java pada tahun 1996 sudah merupakan versi rilis dan bukan versi beta, sehingga diberi nama Java versi 1.0. Pada versi ini terdapat banyak paket standar awal yang terus dikembangkan pada versi berikutnya. Paket-paket yang ada diantaranya:

1. Java.lang: untuk kelas elemen-elemen dasar.
2. Java.io: untuk kelas input dan output, termasuk penggunaan berkas.
3. Java.util: untuk kelas pelengkap, seperti kelas struktur data dan kelas penanggalan.
4. Java.net: untuk kelas TCP/IP, yeng memungkinkan berkomunikasi dengan komputer lain menggunakan jaringan TCP/IP.
5. Java.awt: kelas dasar untuk aplikasi antarmuka dengan pengguna (GUI).
6. Java.applet: kelas dasar aplikasi antarmuka untuk diterapkan pada penjelajah web

Bahasa pemrograman Java dibuat dengan memperhatikan beberapa prinsip sebagai berikut:

1. Harus sederhana, berorientasi objek dan familier.
2. Kuat dan aman.
3. Tidak tergantung platform dan portabel.
4. Bisa dieksekusi dengan performa tinggi.
5. Bisa diinterpretasikan, *threaded* dan dinamis.

Aplikasi-aplikasi berbasis Java akan dikompilasi ke dalam p-code (*bytecode*) dan dapat dijalankan pada berbagai mesin virtual Java (JVM). Dilihat dari sisi penggunaannya, Java merupakan bahasa pemrograman yang bersifat umum/non-spesifik (*general purpose*). Versi asli dan *reference*, vm, dan pustaka class dari Java awalnya dirilis oleh Sun Microsystems di bawah lisensi proprietary. Tetapi sejak bulan Mei tahun 2007, supaya sesuai dengan proses open source Java, maka Sun Microsystems melisensikan ulang teknologi Java dengan teknologi General Public License (GNU). Ada banyak pihak lain yang ikut mengembangkan implementasi alternatif dari teknologi Sun Microsystems, termasuk GNU Compiler for Java (kompiler untuk *bytecode*), GNU Classpath (pustaka standar), dan IcedTea-Web (*plugin browser* untuk applet).

Java dirancang untuk memanfaatkan dependensi implementasi seminimal mungkin. Oleh karena itu aplikasi Java mampu berjalan di beberapa platform sistem operasi yang berbeda. Java dikenal dengan slogannya "*Write Once, Run Anywhere*" (Tulis sekali, Jalankan dimana saja). Yang artinya, program Java hanya perlu ditulis sekali saja, dan bisa dijalankan diberbagai sistem operasi seperti Linux dan Windows tanpa menulis ulang program lagi.

Gambar 2.2 Simbol Bahasa Java

2.2 Java API, JDK, JRE, dan JVM

2.2.1 Java API (*Application Program Interface*)

Jika bahasa pemrograman dipandang sebagai pikiran perangkat lunak, dan JVM sebagai jantung yang membuat perangkat lunak itu terus berdetak,

maka Antarmuka Program Aplikasi (API) pastilah merupakan tangan dan kaki Java.

API merupakan sebuah layer yang berisi class-class yang sudah didefinisikan dan antarmuka pemrograman yang akan membantu para pengembang aplikasi dalam perancangan sebuah aplikasi. API berguna untuk para pengembang untuk dapat mengakses fungsi-fungsi sistem operasi yang diizinkan melalui bahasa Java. Java API terdiri dari kumpulan library (pustaka) yang digunakan untuk keperluan pemrograman. Dengan adanya API, tidak diharuskan membuat program dari awal.

APl menyediakan rangkaian kelas dan komponen yang kaya yang memungkinkan Java melakukan pekerjaan nyata, seperti:

1. Membaca dari dan menulis ke file di hard drive lokal.
2. Membuat antarmuka pengguna grafis dengan menu, tombol, bidang teks, dan daftar *drop-down*.
3. Menggambar gambar dari grafik primitif seperti garis, lingkaran, kotak, dan elips.
4. Mengakses sumber daya jaringan, seperti situs Web atau server jaringan.

5. Menyimpan data dalam struktur data seperti daftar tertaut dan larik.
6. Memanipulasi dan memproses data seperti teks dan angka.
7. Mengambil informasi dari *database* atau memodifikasi catatan.

Karena perpustakaan Java API hanyalah kode yang dapat digunakan kembali, mereka juga ada dalam definisi kelas. Dengan demikian, Java API adalah kumpulan kelas yang besar untuk membuat tugas pengembangan pemrograman lebih produktif. Seringkali, program kita tidak perlu mengimplementasikan struktur data yang dibutuhkannya. Sebagai gantinya, kode dalam API dapat digunakan, atau digunakan kembali dengan spesialisasi melalui pewarisan untuk memenuhi kebutuhan aplikasi kustom kami.

API digunakan dalam berbagai cara umum seperti:

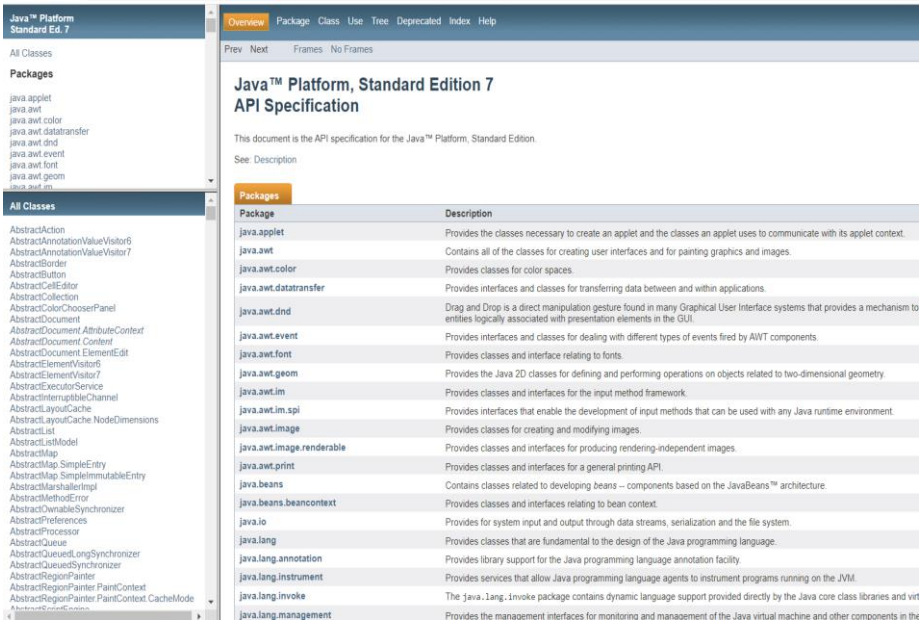
1. Cara paling sederhana dalam menggunakan API adalah dengan membuat instance kelas API; misalnya, untuk membaca file, kita membuat instance FileInputStream. Metode read() digunakan pada objek yang dihasilkan untuk membaca konten file, dan close() yang sama akan melakukan pembersihan yang diperlukan untuk sumber daya sistem seperti deskriptor file untuk digunakan kembali.

2. Kelas baru dapat didefinisikan berdasarkan kelas API dengan pewarisan. Ini memfasilitasi penggunaan kembali kode generik: misalnya, untuk membuat *thread* baru untuk aplikasi *multithread* tertentu, kami mendefinisikan kelas baru berdasarkan kelas Thread, tetapi dengan definisi baru yang relevan dengan aplikasi.
3. Seringkali, variabel kelas dari kelas API dapat digunakan secara langsung tanpa inisialisasi eksplisit, yaitu, *out* adalah variabel kelas publik di kelas System dan dapat digunakan secara langsung untuk tujuan pencetakan ke aliran keluaran standar. Ini adalah bagaimana System.out.println() digunakan.

Karena ada sejumlah besar kelas di Java API sangat bermanfaat untuk mengatur dan mengelompokkannya sesuai dengan fungsinya. Pada JDK 1.5, Java API dasar diatur ke dalam enam paket antara lain: java.lang, java.io, java.util, java.net, java.awt dan java.applet.

1. Paket java.lang terdiri dari kelas java yang penting untuk eksekusi program Java, misalnya, kelas Thread dan System milik paket java.lang.

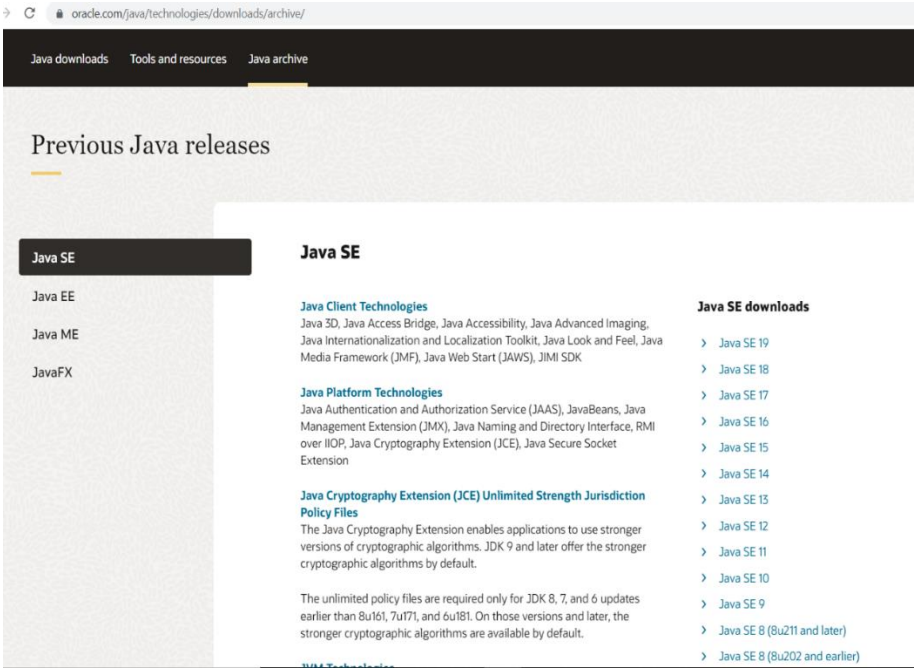
2. Paket java.io terdiri dari kelas-kelas Java yang digunakan untuk fasilitas input dan output, misalnya kelas FileInputStream yang disebutkan sebelumnya termasuk dalam paket java.io.
3. Paket java.net terdiri atau kelas Java yang relevan dengan jaringan, misalnya, kelas Socket milik paket java.net dan digunakan untuk koneksi jaringan ke host di mesin lain.
4. Paket java.util terdiri dari kelas Java untuk fungsionalitas umum seperti kumpulan daftar, representasi tanggal, misalnya kelas Vector dan Date termasuk dalam paket java.util.
5. Paket java.awt terdiri dari kelas Java yang mengimplementasikan *Abstract Windowing Toolkit*. Kelas-kelas ini digunakan untuk membuat antarmuka grafis untuk lingkungan berbasis Windows.
6. Paket java.applet terdiri dari kelas java yang digunakan untuk mendukung eksekusi applet dalam konteks *browser* Web.



Gambar 2.3 Java API (https://docs.oracle.com/javase/7/docs/api/)

2.2.2 JDK (*Java Development Kit*)

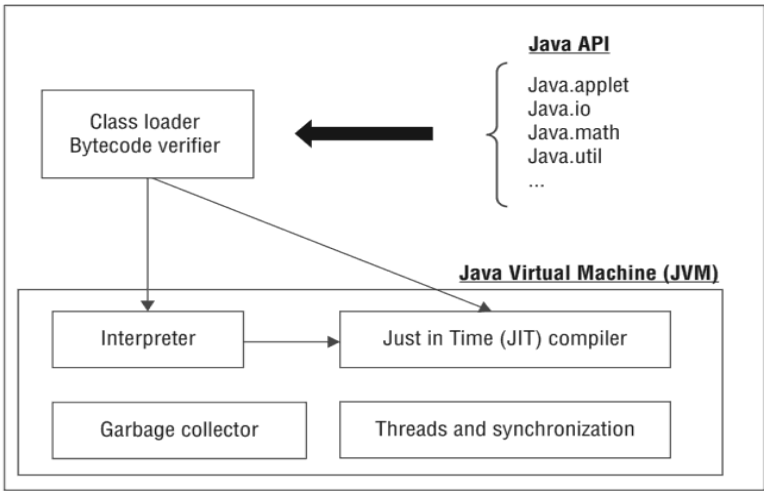
JDK adalah lingkungan pemrograman untuk kompilasi, debugging, dan menjalankan applet Java, aplikasi, dan Java Beans. JDK menyertakan JRE dengan tambahan dari Bahasa Pemrograman Java dan alat pengembangan tambahan serta *tool* API. JDK Sun mendukung Linux, Solaris, dan Microsoft Windows (2000, XP, Vista, dan sampai sekarang). JDK dapat diunduh di website Oracle: https://www.oracle.com/java/technologies/downloads/archive/. Untuk melihat daftar versi JDK seperti pada Gambar 2.4 berikut ini.



Gambar 2.4 Daftar versi JDK (Java SE Developmen Kit)

2.2.3 JRE (*Java Runtime Environment*)

JRE adalah lingkungan perangkat lunak di mana program Java dijalankan. Ini terdiri dari berbagai komponen, seperti yang digambarkan pada Gambar 2.5. JRE juga dikenal sebagai Java Runtime, dan merupakan bagian dari *Java Development Kit* (JDK). JDK adalah seperangkat alat pemrograman yang diperlukan untuk mengembangkan aplikasi Java. JRE terdiri dari library Java, JVM, dan komponen lain yang penting untuk menjalankan aplikasi Java.



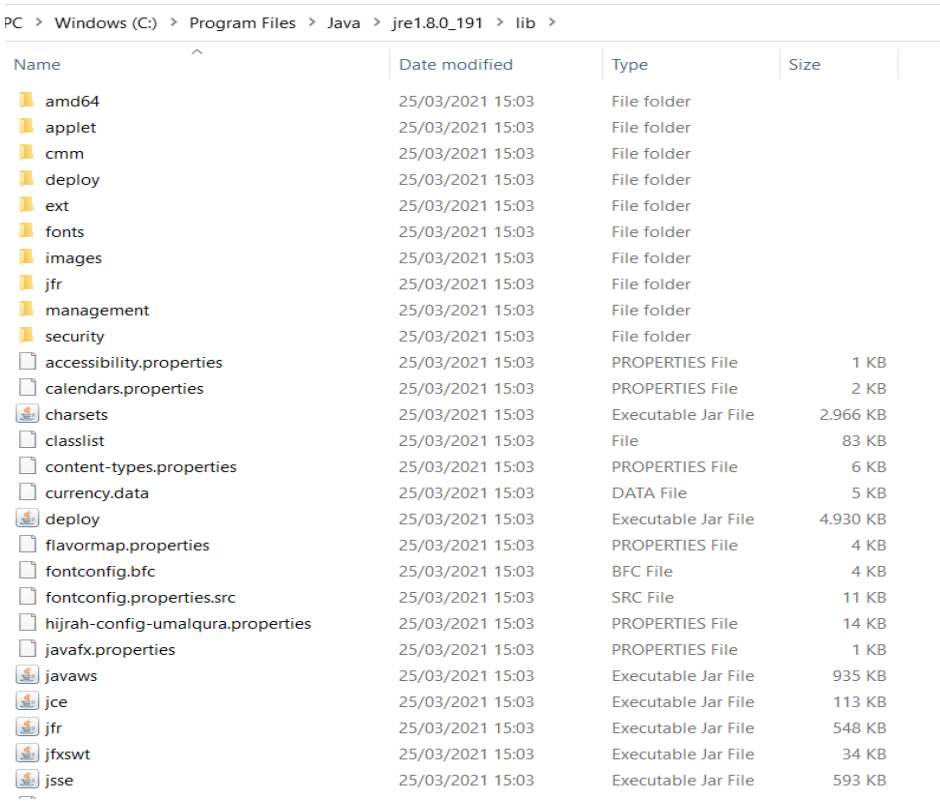
Gambar 2.5 Java Runtime Environment

1. *Class Loader*

   Class loader menempatkan dan membaca file `*.class` yang diperlukan untuk menjalankan program Java dan memuat *bytecode* ke dalam memori. Untuk mengamankan eksekusi yang aman, ia dapat menetapkan bagian memori yang berbeda (*namespace*) ke kelas yang diperoleh secara lokal versus dari jarak jauh. Kelas biasanya dirakit menjadi perpustakaan yang disimpan secara fisik dalam file JAR (*Java Archive*). Pustaka mungkin telah ditulis oleh pengguna atau diperoleh secara eksternal. Untuk menemukan kelas, *class loader* pertama-tama akan menemukan pustaka yang sesuai dan memuat kelas sesuai kebutuhan program (disebut pemuatan

sesuai permintaan). *Class loader* pada dasarnya memiliki tiga subkomponen yaitu: *Bootstrap class loader, Extensions class loader,* dan *System class loader*.

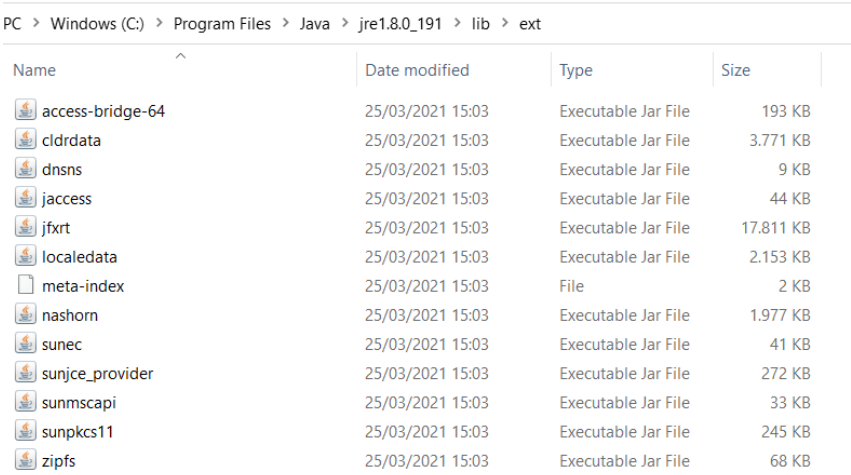
Bootstrap class loader memuat pustaka inti Java yang terletak di `<JAVA_HOME>/jre/lib`. Isi direktori ini dalam lingkungan JRE8 ditunjukkan pada Gambar 2.6.



Gambar 2.6 Direktori JRE8

Dapat dilihat file `*.jar` berbeda yang akan dipertimbangkan (charsets.jar, deploy.jar, dan sebagainya).

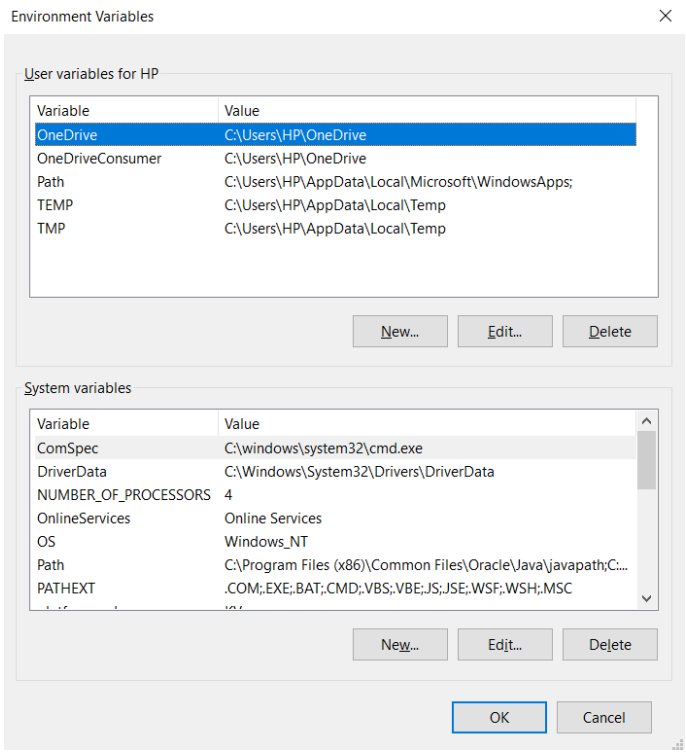
Extensions class loader memuat kelas dari direktori ekstensi `<JAVA_HOME>/jre/lib/ext`. Gambar 2.7 menunjukkan isi direktori ekstensi di lingkungan JRE8.



Gambar 2.7 *Extensions class loader*

Sedangkan *System class loader* memuat kode dari lokasi yang ditentukan dalam variabel lingkungan CLASSPATH, yang ditentukan oleh sistem operasi. Yang terakhir menyediakan jalur ke semua direktori fisik tempat *system class loader* dapat mencari file Java. Ini dapat ditemukan di Windows dengan masuk ke Advanced System Settings -> tab Advanced ->

Environment Variables. Gambar 2.8 menunjukkan jendela dari sistem operasi Windows 10.



Gambar 2.8 *System class loader*

2. *Bytecode Verifier*

Bytecode verifier (pemverifikasi *bytecode*) memeriksa untuk memastikan *bytecode* valid tanpa melanggar aturan keamanan Java. Ini memberikan perhatian khusus untuk memeriksa semua variabel dan ekspresi dalam kode dan memastikan bahwa tidak ada akses tidak sah ke memori. Perhatikan bahwa ketika program Java dipanggil, kita dapat memilih untuk

menonaktifkan *bytecode verifier* (yang akan membuatnya berjalan sedikit lebih cepat), mengaktifkannya hanya untuk kode yang diunduh dari jarak jauh dari jaringan, atau mengaktifkannya untuk semua kode. Setelah kode diverifikasi, itu akan ditawarkan ke Java Virtual Machine (JVM) untuk interupsi.

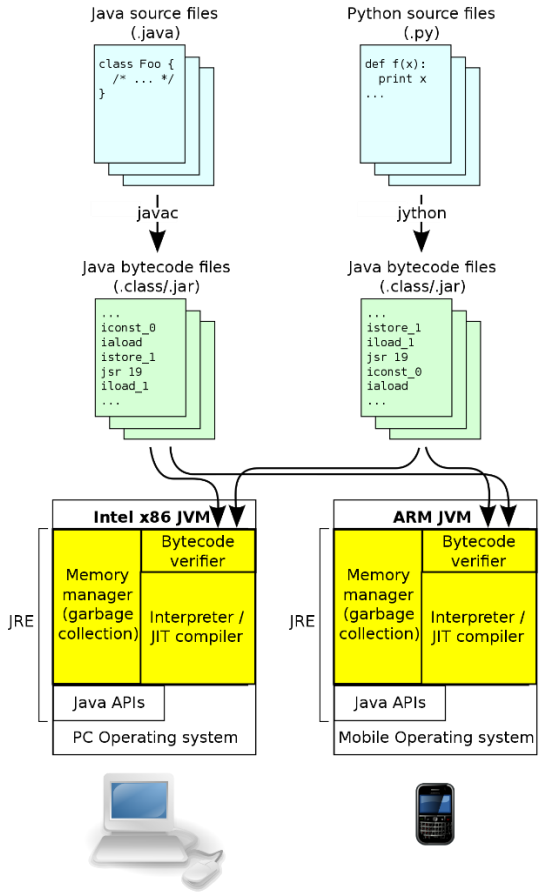
2.2.4 *Java Virtual Machine* (JVM)

Java Virtual Machine (JVM) dapat dianggap sebagai mesin komputer abstrak yang mampu mengeksekusi bytecode pada platform perangkat keras tertentu. Ini merupakan inti dari filosofi "tulis sekali, jalankan di mana-mana". Berbagai implementasi JVM telah disediakan untuk berbagai perangkat keras dan/atau lingkungan sistem operasi. JVM yang paling populer adalah HotSpot yang diproduksi oleh Oracle. Ini tersedia untuk Windows, Linux, Solaris, dan Mac OS X. Komponen kunci dari JVM adalah interpreter yang bertanggung jawab untuk menginterpretasikan instruksi bytecode. Pengumpul sampah membersihkan memori yang tidak digunakan untuk meningkatkan efisiensi program. JVM biasanya juga mencakup fasilitas untuk multithreading dan sinkronisasi, di mana program Java dapat dieksekusi dalam satu atau lebih jalur eksekusi paralel (utas) yang

dijadwalkan pada satu atau lebih CPU, dengan ini secara signifikan mempercepat waktu eksekusinya.

Mesin Virtual Java adalah landasan platform Java. Ini adalah komponen teknologi yang bertanggung jawab atas independensi perangkat keras dan sistem operasinya, ukuran kecil dari kode yang dikompilasi, dan kemampuannya untuk melindungi pengguna dari program jahat.

Java Virtual Machine adalah mesin komputasi abstrak. Seperti mesin komputasi nyata, ia memiliki set instruksi dan memanipulasi berbagai area memori pada saat dijalankan. Cukup umum untuk mengimplementasikan bahasa pemrograman menggunakan mesin virtual; mesin virtual yang paling terkenal mungkin mesin P-Code dari UCSD Pascal.



Gambar 2.9 Arsitektur JVM

2.3 Karakteristik Java

Masing-masing bahasa pemrograman memiliki karakter tersendiri. Meskipun banyak macam bahasa pemrograman yang lain, bahasa Java masih banyak diminati, salah satunya dikarenakan bahasa Java yang portabilitasnya mudah dilakukan. Berikut ini beberapa karakter dari bahasa Java antara lain:

1. Java *is Simple*

   Bahasa Java merupakan bahasa pemrograman yang dikembangkan dari bahasa C++, tetapi dibuat lebih sederhana dan lebih ditingkatkan lagi kemampuannya. Misal, pada Java tidak ada lagi *pointer* dan *multiple inheritance* yang membuat pemrograman menjadi lebih rumit. *Multiple inheritance* pada Java digantikan oleh *interface* (antarmuka) dan *pointer* dihilangkan.

2. Java *is Object Oriented*

   Pemrograman berorientasi objek merupakan suatu metodologi perancangan program berdasarkan objek, dimana semua hal dapat dianggap sebagai objek, misal manusia, tanaman, komputer, dan bahkan basis data. Kelebihan dari pemrograman berorientasi objek ialah sangat fleksibel, modular, penggunaan kembali kode program melalui enkapsulisasi data, *inheritance*, dan *polymorpishm*.

3. Java *is Distributed*

   Distributed computing adalah metode komputerisasi dengan menggunakan beberapa komputer yang dihubungkan dengan jaringan untuk mengelola tugas-tugas tertentu. Java memiliki kemampuan

networking yang baik, misal untuk mengirim dan menerima data dari sebuah file.

4. Java *is Interpreted*

   Java adalah bahasa yang menggunakan interpreter atau penerjemah agar dapat menjalankan program, maka pada komputer tujuan harus ada interpreternya. Hal tersebut dikarenakan interpreter Java menerjemahkan kode *bytecode* ke dalam bahasa mesin dari komputer tujuan.

5. Java *is Robust*

   Robust memiliki arti dapat diandalkan. Java telah menghilangkan dan mengantisipasi berbagai macam gangguan (*bug*) dan kesalahan-kesalahan yang umum terjadi. Misal dengan menghilangkan *pointer*, maka menghilangkan kemungkinan adanya *overwriting* memori dan menghasilkan data yang tidak utuh. Pada Java untuk menuliskan kode program menggunakan *exception* (penanganan kesalahan sehingga program tidak hang ketika ada kesalahan).

6. Java *is Secure*

   Sebagai bahasa pemrograman internet, Java digunakan pada lingkungan *networking* dan terdistribusi. Jika mengunduh Java Applet

dan menjalankannya pada komputer, maka tidak perlu khawatir tentang kerusakan yang mungkin akan disebabkan olehnya. Hal ini dikarenakan Java tidak menyediakan akses secara bebas ke sistem secara langsung.

7. Java *is Architecture-Neutral*

   Program yang dihasilkan oleh Java tidak tergantung kepada arsitektur komputer tertentu. Karena program Java dapat berjalan dalam lingkungan JVM (*Java Virtual Machine*), sehingga dapat dijalankan pada arsitektur komputer yang berbeda. Hanya diperlukan menginstal JVM yang tepat untuk masing-masing platform yang dituju.

8. Java *is Portable*

   Karena Java bersifat netral terhadap arsitektur komputer dan sistem operasi, Java dapat dibawa dan dijalankan dimana-mana, program Java dapat dikompilasi dan menjalankannya pada mesin yang lain tanpa melakukan kompilasi ulang.

9. Java *Performance*

   Kinerja Java sering mendapat kritikan atau dianggap lambat, hal ini disebabkan oleh program yang dijalankan melalui JVM. Namun

dengan adanya teknologi prosesor yang memiliki kecepatan proses tinggi, kelemahan ini dapat diatasi.

10. Java *is Multi-Thread*

    Sama halnya dengan bahasa pemrograman lainnya yang mengenal pemrosesan beberapa tugas secara bersamaan (*multi-threading*). Misal, pada saat mengunduh sebuah file video dapat sambil memainkan bagian file yang telah terunduh, pada Java kemampuan ini sudah *build-in*, jadi tidak diperlukan pengaktifan dengan cara memanggil prosedur tertentu.

11. Java *is Dynamic*

    Java telah dirancang untuk dapat beradaptasi pada lingkungan yang selalu berubah-ubah. Misal, dapat dilakukan *load* sebuah *class* secara langsung tanpa melakukan rekompilasi ulang. Dengan hal ini para pengembang aplikasi tidak perlu membuat dan bagi pengguna menginstal *software* versi baru jika ada penambahan fitur dapat ditambahkan langsung.

2.4 Komponen Java

Dalam bahasa pemrograman terdapat komponen-komponen program yang membangun struktur-struktur dan membentuk sebuah program. Komponen-komponen tersebut antara lain:

1. *Identifier*

   Setiap entitas di dunia memiliki nama atau identitas, dalam dunia pemrograman identitas disebut sebagai *identifier*. *Identifier* diberikan kepada *variabel, class, method, packages, constans*. *Identifier* merupakan simbol berupa deretan karakter yang tersusun memiliki arti tertentu. Dalam menuliskan identifier terdapat aturan tertentu sebagai berikut:

   a. Terdiri dari deretan karakter yang terdiri dari huruf, angka, *underscore* (_), dan simbol dollar ($).
   b. Harus dimulai oleh huruf, *underscore* (_), atau simbol dollar ($). Tidak dapat dimulai dengan angka.
   c. Tidak boleh mengambil dari kata kunci.
   d. Tidak dapat berupa nilai *boolean* (*true, false* atau *null*).
   e. Panjangnya bervariasi tanpa batas tertentu.

2. Variabel

   Variabel digunakan untuk menyimpan data di dalam program. Pendeklarasian variabel dengan cara menyebutkan dahulu tipe datanya kemudian identifier yang diikuti tanda titik koma (;).

   Bentuk umum penulisan: tipedata namavariabel;

   Contoh:

```
int jumlah;
int total, total2;
```

3. Konstanta

   Konstanta juga digunakan sebagai penampung data, namun tidak dapat diubah nilainya ketika program berjalan. Konstanta nilainya ditentukan saat program dirancang. Bentuk umum penulisan: `final tipedata namakonstanta = nilai;`

   Final adalah kata kunci di dalam Java yang berarti bahwa konstanta tersebut tidak dapat diubah nilainya. Contoh:

```
final double PI = 3.14;
```

4. *Assignment Operator*

   Merupakan operator penugasan berupa tanda sama dengan (=). Operator ini berfungsi memberikan nilai kepada variabel atau

konstanta. Misal sebuah variabel dideklarasikan dan diberi nilai awal, dengan penulisan seperti ini: `int angka = 7;`. Selain itu nilai yang diberikan tidak hanya sebuah nilai secara langsung, namun dapat berupa rumus seperti ini: `luas_persegi = p*l;`

5. Tipe Data Numerik

   Merupakan tipe data yang menampung nilai numerik (angka). Pada gambar 2.10 merupakan beberapa tipe data numerik yang ada pada Java.

Tipe Data	Jangkauan	Alokasi memori
Byte	-2^7(-128) s/d $2^7 - 1$(127)	8 bit signed
Short	-2^{15}(-32768) s/d $2^{15} - 1$(32767)	16 bit signed
Int	-2^{31}(-2147483648) s/d $2^{31} - 1$(2147483647)	32 bit signed
Long	-2^{63} s/d 2^{63}(atau -9223372036854775808 s/d 9223372036854775808)	64 bit signed
Float	-3.4E38 s/d 3.4E38 (akurasi 6 – 7 digit)	32 bit IEEE 754
Double	-1.7E308 s/d 1.7E308 (akurasi 14 – 15 digit)	64 bit IEEE 754

Gambar 2.10 Tipe Data Numerik

6. Operator Numerik

   Operator numerik digunakan untuk menangani pengolahan tipe data numerik. Java menyediakan operator numerik dengan simbol-simbol, dimana ketika dibaca oleh kompiler akan diartikan sebagai tindakan

untuk melakukan pengolahan terhadap data numerik. Berikut gambar 2.11 merupakan operator numerik yang ada pada Java.

Operator	Kegunaan	Contoh
+	Melakukan penjumlahan	X = 12 + 3
-	Melakukan pengurangan	Y = 12-3
*	Melakukan perkalian	Z = 30000 * 3000
/	Melakukan pembagian	A = 6/3
%	Digunakan untuk mencari sisa dari hasil bagi dua bilangan, nama operator ini adalah modulus.	B = 20 % 3 Hasilnya 2

Gambar 2.11 Operator Numerik

Pada Java dapat dilakukan pendefinisian tipe data sekaligus menetapkan operasi numeriknya jika ternyata variabel yang digunakan untuk menampung hasil dari perhitungan. Operator-operator yang tersebut pada gambar di atas menggunakan dua buah operan yaitu variabel atau numerik literal (bilangannya langsung) yang ada di sebelah kanan atau kiri operator.

Contoh pedeklarasian tipe data disertai dengan variabel dan menggunakan operator sekaligus adalah seperti berikut.

```
int a = 10+3;
double x = 30.0 - 0.1;
int b = 1/2;
```

7. Urutan Operator

   Pada operasi matematika terdapat urutan pengerjaan operator-operator misalnya operator kali dan bagi dikerjaan terlebih dahulu daripada tambah dan kurang. Urutan pengerjaan mulai dari tanda kurung ( ), karena operator yang ada di dalam tanda kurung akan dikerjakan terlebih dahulu dengan pengerjaan dari sebalah kiri ke kanan. Harus diperhatikan dengan benar hal ini, karena salah menyusun urutan pengerjaan akan mempengaruhi hasil (kesalahan logika).

8. Operator Penyingkat

   Dalam pemrograman akan dijumpai penggunaan contoh kode seperti ini `x = x+1;`. Kode tersebut biasanya terdapat pada sebuah loop atau perulangan dengan fungsi untuk melakukan perhitungan berapa kali *loop* telah berjalan (*counter*). Pada gambar 2.12 merupakan beberapa operator penyingkat pada Java.

Operator	Kegunaan	Contoh.
+=	Melakukan penambahan	K += 1 Sama dengan : K = K + 1
-=	Melakukan pengurangan	F -= 1 Sama dengan : F = F-1
*=	Melakukan perkalian	J *= 2 Sama dengan : J = J * 2
/=	Melakukan pembagian	X /= 2 Sama dengan : X = X/2
%=	Mencari sisa pembagian.	Z %= 5 Sama dengan : Z = Z % 5

Gambar 2.12 Operator Penyingkat

9. Konversi Tipe Data Numerik

Terkadang dalam sebuah program diperlukan sebuah perhitungan yang mencampurkan tipe data numerik yang tidak sama ke dalam sebuah variabel yang berbeda juga. Dalam hal ini Java memberikan kemudahan antara lain:

a. Jika ada sebuah operan bertipe double, maka yang lain juga dikonversi ke double.
b. Jika salah satu dari operan bertipe float, maka yang lain juga dikonversi ke float.
c. Jika salah satu dari operan tipe datanya long, maka yang lain juga dikonversi ke long.

d. Jika tidak, maka kedua operan akan dikonversi ke integer.

Misal hasil dari ½ adalah 0 (karena keduanya integer, dan hasil pembagian integer merupakan nilai tanpa desimal), sedangkan hasil yang benar dari ½ adalah0.5.

10. Tipe Data Char dan Operatornya

Tipe data ini menggunakan *keyword* char untuk mendeklarasikan tipe datanya. Nilai yang dapat ditampung berupa karakter huruf atau angka atau karakter lainnya. Pemberian nilai ke dalam variabel bertipe char harus disertai dengan tanda kutip tunggal (''), misal seperti ini: `char huruf = 'A'; char angka = '2';`

Yang perlu diperhatikan adalah penggunaan tanda petik. Tanpa petik tunggal digunakan untuk memberi nilai ke tipe data char, sedangkan tanda petik ganda digunakan untuk memberi nilai ke tipe data string.

11. Konversi Tipe Data Char ke Numerik

Tipe data char dapat diubah ke tipe data numerik, misal integer. Ketika sebuah tipe data integer diubah ke tipe data char, hanya 16 bit data yang digunakan, bagian yang lainnya diabaikan. Sedangkan ketika sebuah tipe data pecahan (float) diubah ke tipe data char, maka tipe data tersebut diubah terlebih dahulu ke tipe data integer,

kemudian diubah ke tipe data char. yang menjadikan perubahan tipe data ke tipe data lainnya menjadi error adalah tidak diperhatikannya ukuran dari alokasi tipe data pada memori.

12. Tipe Data Boolean dan Operatornya

Merupakan tipe data yang hanya memiliki kemungkinan nilai yaitu *true* (benar) atau *false* (salah). Tipe data ini biasanya digunakan sebagai penampung ekspresi-ekspresi perbandingan yang hasilnya memberikan nilai benar atau salah. Karena tipe data ini banyak digunakan untuk perbandingan, maka terdapat operator-operator perbandingan yang dapat dilakukan pada Java. Pada gambar 2.13 berikut ini adalah operator-operator perbandingannya.

Operator	Nama	Contoh	Jawaban
<	Lebih kecil	1 < 2	True
>	Lebih besar	2 > 1	True
==	Sama dengan	2 == 2	True
<=	Lebih kecil atau sama dengan	2 <= 2, 2 <= 3	True, false
>=	Lebih besar atau sama dengan	2 >= 2, 3 >= 2	True, false
!=	Tidak sama dengan	'A' != 'C'	True

Gambar 2.13 Operator Perbandingan

Tipe data boolean juga memiliki operator sendiri dan tipe data yang dihasilkan dari operasinya adalah boolean juga. Gambar 2.14 berikut ini adalah operator boolean yang ada pada Java.

Operator	Nama	Keterangan
!	Not	Negasi (tidak)
&&	And	Konjungsi
||	Or	Disjungsi

Gambar 2.14 Operator Boolean

13. Tipe Data String

Merupakan tipe data yang dapat menampung urutan karakter yang memiliki arti tersendiri. Urutan karakter berupa kata atau kalimat. Pendeklarasian variabel string seperti ini: `String text = "Hallo Java!";`, selain itu juga dapat dilakukan penggabungan string dengan cara seperti ini: `String text = "Hallo Java!" + "Nama saya Java";`.

14. Konversi Tipe Data String ke Numerik

Cara untuk mengubah tipe data string menjadi numerik ialah dengan menggunakan *method parseInt* dari *class* Integer.

```
int nilai_string =
Integer.parseInt(string_int);
```

sedangkan untuk konversi ke tipe data double menggunakan *method parseDouble* di class double.

```
double nilai_double = Double.parseDouble
(string_double);
```

Kedua class Integer dan class Doule tersebut terdapat pada package java.lang.

15. Pesan *Error* dan *Debugging*

    Dalam Java terdapat beberapa jenis kesalahan yaitu *syntax error, runtime error*, dan *logic error*.

    a. *Syntax error*

       Error yang muncul selama proses kompilasi disebut *syntax error* atau *compilation error*. Kesalahan ini biasanya berkaitan dengan konstruksi kode saat pengetikan atau berupa salah ketik kata kunci atau pengurangan tanda baca seperti ; atau tanda petik, kurung kurawal, dan lain sebagainya.

       Contoh:

```
JoptionPane.showMessageDialog(null,
“Hallo”, “Hallo Box”,
JoptionPane.INFORMATION_MESSAGE);
```

b. *Runtime error*

Error ini muncul saat program sedang berjalan, biasanya disebabkan karena program tidak mampu melakukan perintah tertentu karena sebab tertentu. *Error* ini mengakibatkan program tidak dapat dilanjutkan dan diakhiri dengan segera. Biasanya *error* jenis ini berasal dari masukkan (*input*) yang diberikan oleh pengguna yang tidak tertangani oleh program.

Untuk mengatasi hal tersebut, dapat dideteksi dan memberikan penanganan kesalahan pemasukkan data. *Error* ini juga dapat disebabkan oleh pembagian dengan bilangan 0 (nol), pesan kesalahannya adalah division by zero. Contoh:

```
int x = 10/0;
```

c. *Logic error*

Error ini disebabkan adanya algoritma yang kurang baik saat pembuatan program. Hal ini biasa terjadi pada awal-awal pemrograman. *Error* ini biasanya menyebabkan program tidak bekerja dan tidak menghasilkan luaran (*output*) seperti yang diharapkan.

d. Debugging

Pada dasarnya *syntax error* dan *runtime error* tidak sulit untuk ditemukan dan ditangani dengan segera, berbeda dengan *logic error*. *Logic error* sulit ditemukan, sering disebut juga sebagai *bug*. Sedangkan proses untuk melakukan pencarian dan melakukan pembenaran *error* disebut sebagai proses *debugging*. Cara melakukan *debugging* ada bermacam-macam, mulai dari membaca kode program dari atas hingga kode paling akhir, dan dapat juga melakukan *debugging* dengan menggunakan *debugger utility*.

BAB 3
PERSIAPAN PENGGUNAAN JAVA

Persiapan awal untuk dapat menggunakan bahasa pemrograman Java, maka perlu dilakukan beberapa langkah-langkah antara lain: instalasi *Java Development Kit* (JDK); ekstrak Dokumentasi JDK, dalam hal ini untuk mempelajari Java API; dan instalasi *Integrated Development Environment* (IDE) yang akan digunakan, misal menggunakan NetBeans IDE.

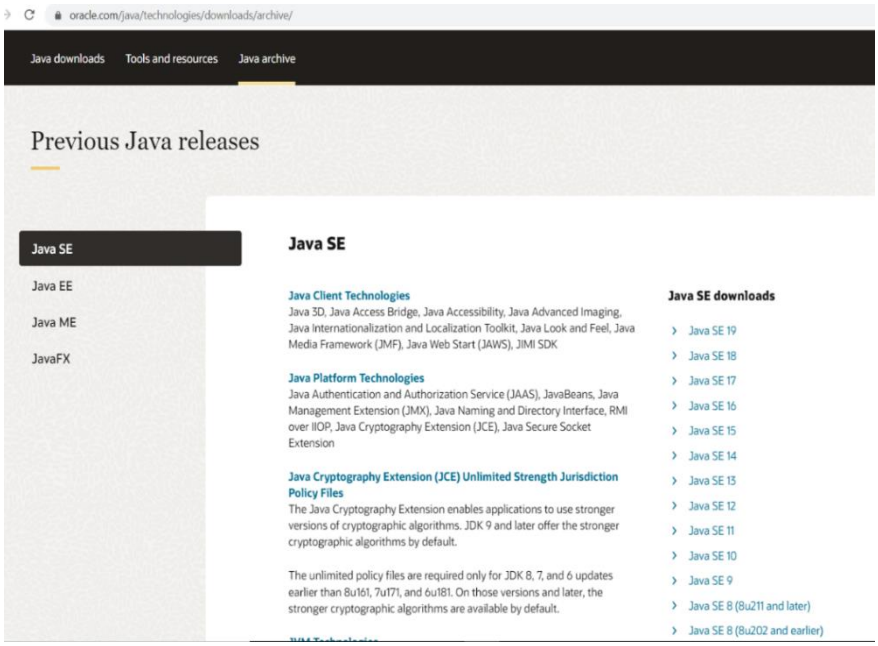
3.1 Instalasi JDK

Dalam penggunaan bahasa pemrograman Java, di dalam komputer harus terinstal dahulu JDK. JDK atau yang pernah disebut dengan *Java Software Development Kit* (Java SDK) merupakan perangkat aplikasi yang digunakan untuk mengembangkan program dengan bahasa pemrograman Java. Di dalam JDK terdapat 3 komponen utama yaitu Java API, Java Virtual Machine, dan Java Compiler atau berupa JRE ditambah dengan Java Compiler.

JDK merupakan perangkat minimal yang harus disiapkan sebelum menggunakan bahasa pemrograman Java. Aplikasi ini dapat diunduh secara gratis melalui situs Oracle

https://www.oracle.com/java/technologies/downloads/archive/. Langkah-langkah untuk instalasi JDK adalah seperti berikut:

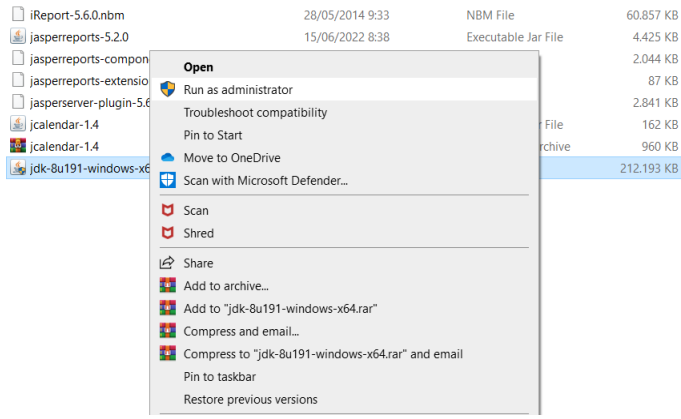
1. Unduh dahulu JDK di situs resmi Oracle seperti gambar di bawah ini.



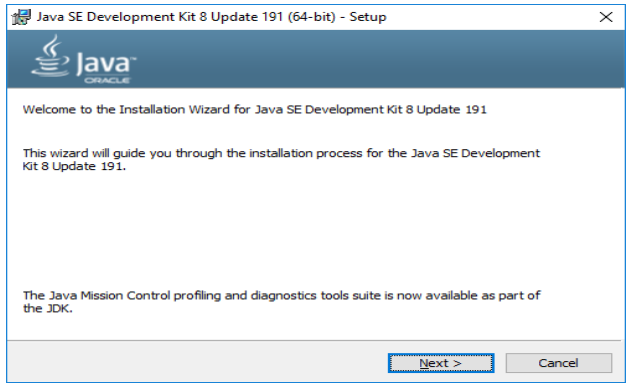
Gambar 3.1 Java Archive pada website Oracle

Dalam website tersebut terdapat banyak pilihan versi dari JDK. Pilihlah yang sesuai dengan spesifikasi komputer yang ada, misalnya disini digunakan JDK 8u191.

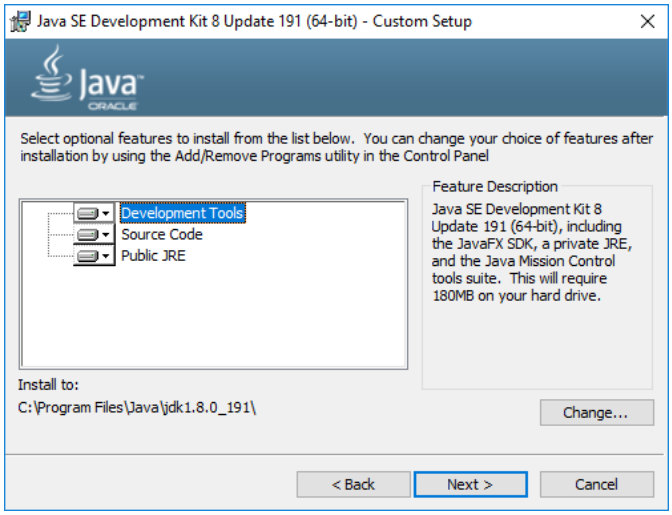
2. Selanjutnya jika sudah terunduh, klik ganda filenya atau dengan klik kanan → Run as administrator



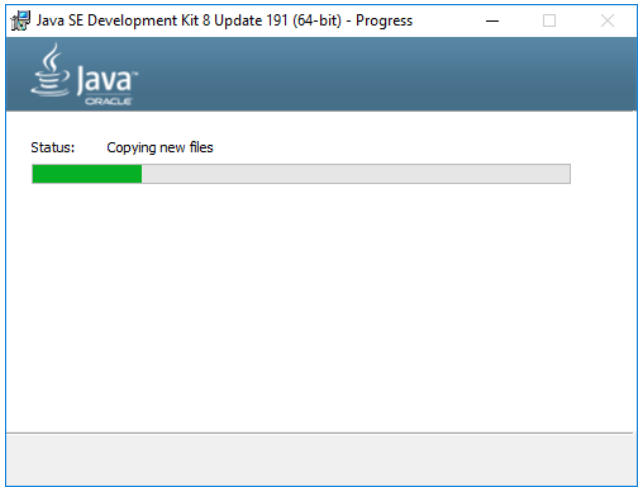
3. Kemudian akan muncul tahapan instalasinya, klik tombol **Next** untuk melanjutkan prosesnya.



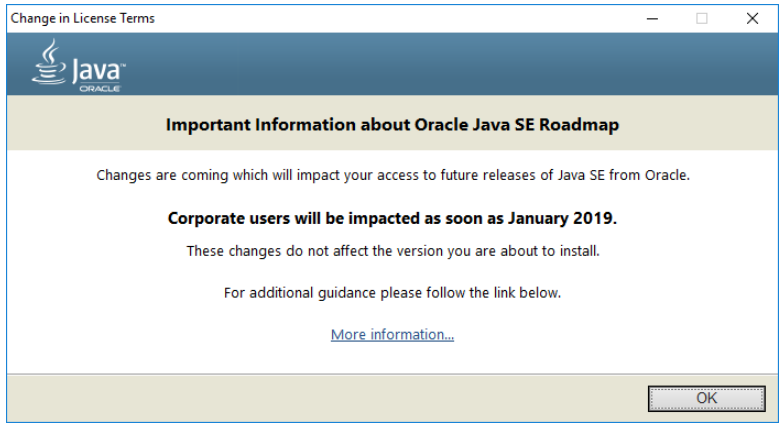
4. Kemudian muncul jendela Custom Setup untuk mengatur lokasi instalasi → klik tombol **Next** untuk melanjutkan proses selanjutnya.



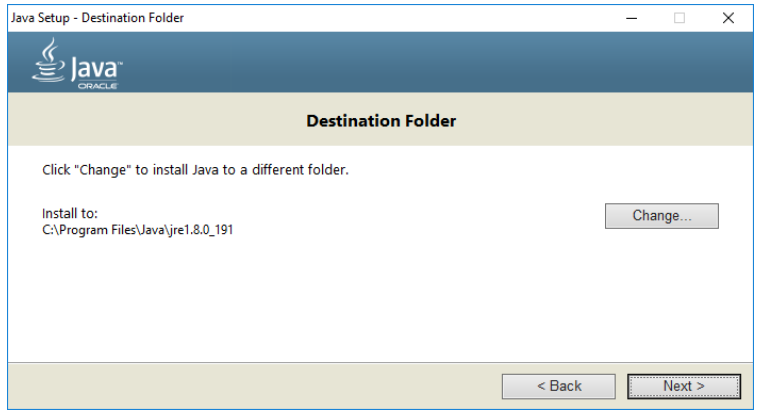
5. Kemudian akan muncul *progress* proses instalasinya.



6. Kemudian muncul jendela Change in License Terms → klik tombol **OK**.

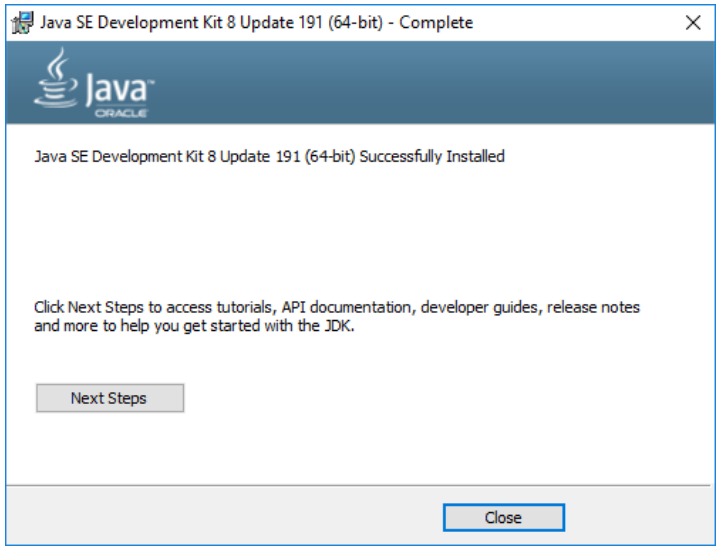


7. Kemudian muncul pemilihan lokasi/folder tujuan instalasi *Java Runtime Environment* (JRE). Untuk mengubah pilih tombol **Change**, jika memilih lokasi default klik tombol **Next**.



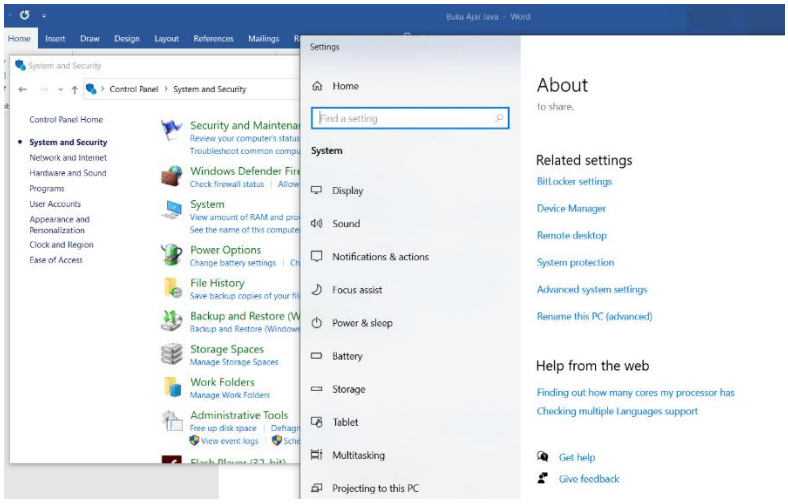
8. Kemudian muncul jendela *progress*.

9. Proses sudah selesai, *Java Development Kit* (JDK) beserta *Java Runtime Environment* (JRE) sudah terinstal di komputer.

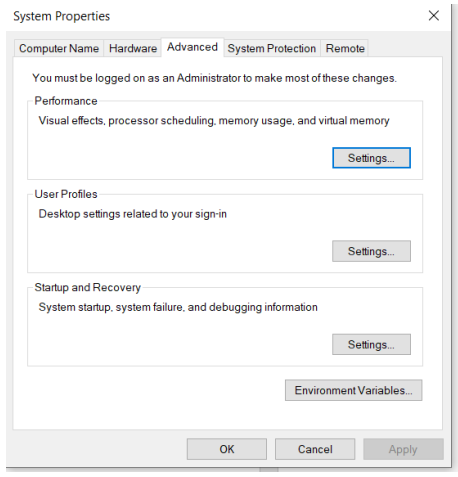


Terdapat dua pilihan tombol yaitu **Next Steps** dan **Close**. Jika memilih tombol **Next Steps** maka akan menampilkan dokumentasi untuk Java. Dan **Close** untuk mengakhiri proses instalasi.

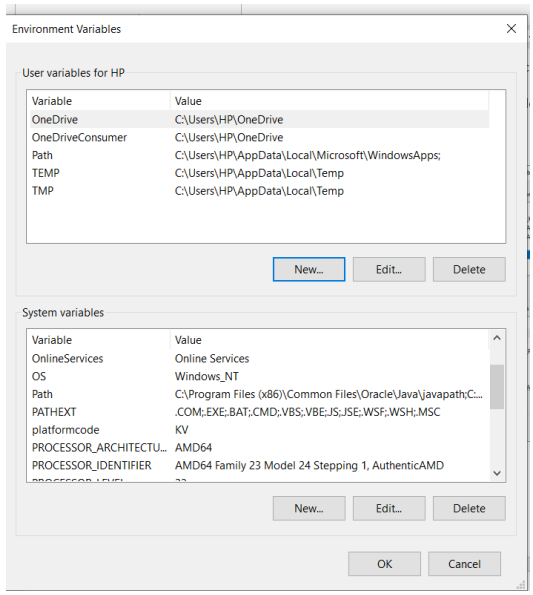
10. Kemudian lakukan pengaturan pada System Properties dengan cara membuka Control Panel → System and Security → System → Related settings → Advanced system settings → maka akan mucul System Properties.

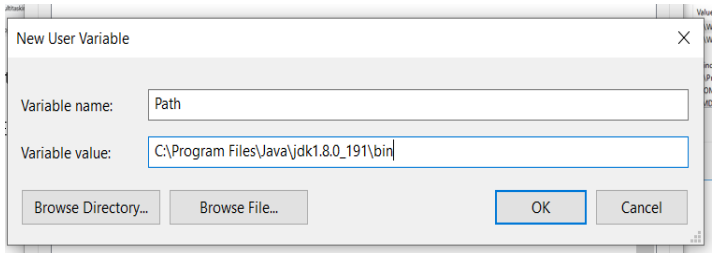


11. Pada jendela dialog System Properties pilih tab menu Advanced → klik tombol Environment Variables.



12. Setelah muncul jendela dialog Environment Variables → klik tombol **New**, buat variabel baru dengan isian seperti berikut.

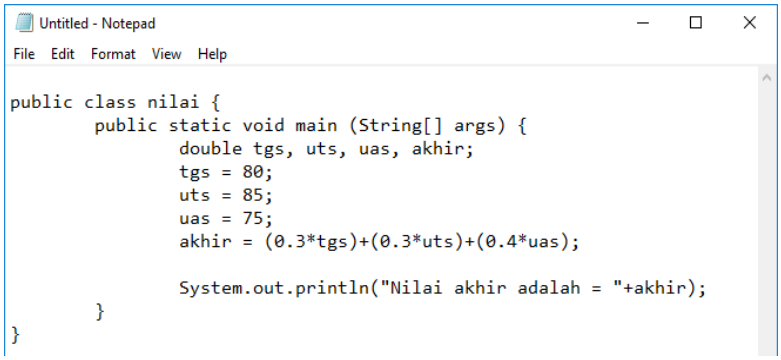


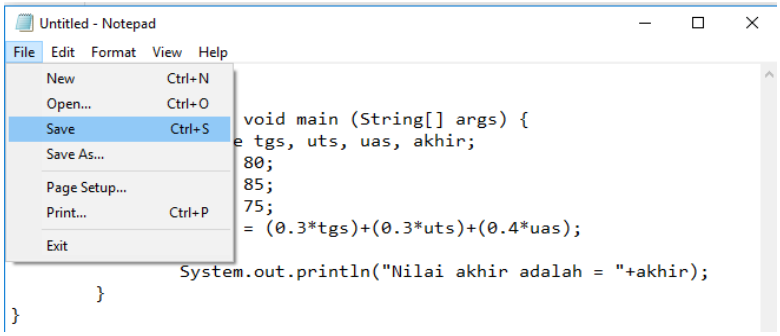


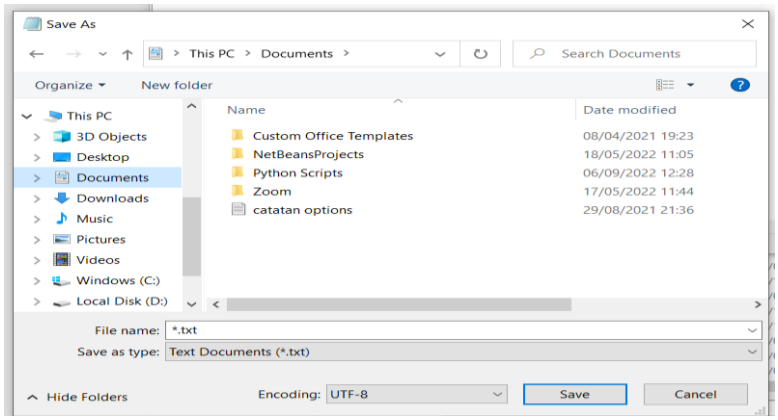
13. Setelah membuat variabel baru, klik **OK** hingga keluar dari System Properties. Kemudian restart komputer. Dengan terinstalnya JDK, komputer telah siap digunakan untuk melakukan kompilasi dan menjalankan program Java.

Setelah melakukan persiapan untuk pemrograman Java, selanjutnya dapat mulai melakukan pembelajaran pemrograman Java dengan menggunakan aplikasi Notepad sebagai teks editor dan Command Prompt untuk menjalankan program yang telah dibuat. Langkah-langkahnya adalah seperti berikut:

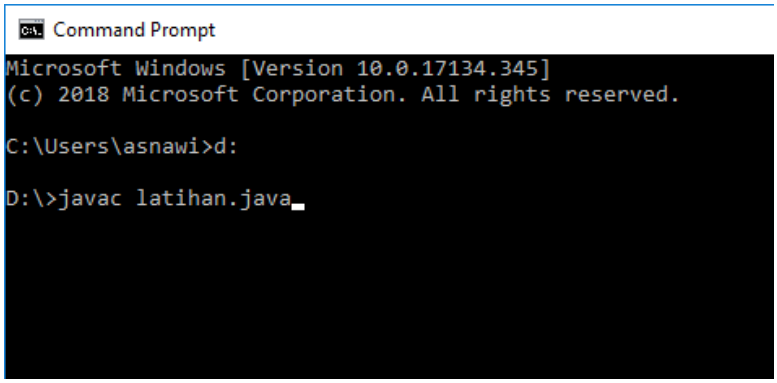
1. Buka aplikasi Notepad
2. Ketikka kode program yang akan dibuat
3. Simpan file dengan cara pilih menu File → Save (atau bisa dengan menekan tombol Ctrl+S) → isikan nama file dengan diakhiri ekstensi **.java** dan atur Save as type menjadi All Files → klik Save.

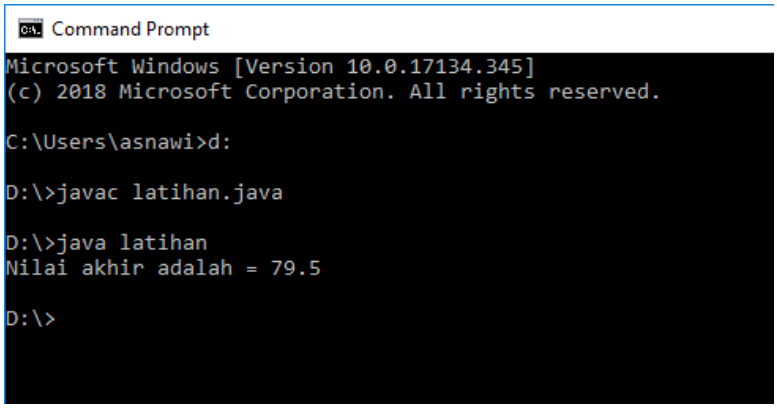






4. Kemudian buka Command Prompt
5. Masuk ke dalam direktori tempat penyimpanan file .java yang telah dibuat
6. Sebelum menjalankan file .java, terlebih dahulu dilakukan kompilasi program Java dengan perintah **javac nama_file.java** → jalankan program Java dengan perintah **java nama_file_tanpa_ekstensi.java**.





7. Selesai

Yang perlu diperhatikan adalah langkah untuk mengkompilasi terlebih dahulu sebelum menjalankan file program .java. Hal ini juga dilakukan ketika menyimpan pembaruan kode dari file sebelumnya, jika tidak dilakukan kompilasi ulang maka yang dijalankan adalah file program yang di awal belum ada perubahan.

3.2 Operator Dasar Java

Operator merupakan simbol operasi yang dapat digunakan pada variabel atau operan yang digunakan ketika membuat sebuah program. Operator dasar pada Java antara lain operator aritmatika, operator logika, operator pembanding, operator penaikan dan penurunan, dan operator penugasan. Operator tersebut memiliki fungsinya masing-masing. Penjelasan lebih lengkapnya seperti berikut ini.

1. **Operator Aritmatika**

Operator ini digunakan untuk melakukan operasi matematika atau membuat sebuah program yang membutuhkan operasi matematika. Tetapi tidak semua simbol yang sama digunakan dalam perhitungan matematika bisa digunakan untuk perhitungan dalam pemrograman. Simbol-simbol yang dapat dimengerti oleh pemrograman antara lain sebagai berikut.

Tabel 3.1 Simbol-simbol Operator Aritmatika

Simbol Matematika	Simbol Aritmatika
+ (penjumlahan)	+ (penjumlahan dan untuk menyambung ***string*****)**
- (pengurangan)	-
: atau / (pembagian)	/
x (perkalian)	*
Modulus / sisa pembagian	%

2. **Operator Logika**

Operator logika sama dengan operator *boolean* pada pemrograman yaitu bernilai benar atau salah (*true or false*). Macam-macam operator logika antara lain seperti berikut ini.

Tabel 3.2 Simbol-simbol Operator Logika

Simbol Operator Logika	Penjelasan
!	Simbol NOT
&&	Simbol AND
||	Simbol OR

Operator-operator tersebut memberikan hasil yang beragam tergantung kondisi atau objek yang digunakan. Hasil atau nilai akhir dari penggunaan operator-operator tersebut seperti berikut.

Tabel 3.3 Contoh Nilai Operator Logika

A	B	Not A	Not B	A and B	A or B
False	False	True	True	False	False
False	True	True	False	False	True
True	False	False	True	False	True
True	True	False	False	True	True

a. NOT bertugas mengecek operan dan membalikkan nilainya (dari *true* ke *false*, dari *false* ke *true*).
b. AND akan menghasilkan nilai *true* hanya jika kedua operan bernilai *true*, selain dari itu maka nilai akhirnya *false*.
c. OR akan mengahasilkan nilai *false* hanya jika kedua operan bernilai *false*, selain dari itu maka nilai akhirnya *true*.

3. Operator Pembanding

Operator pembanding digunakan untuk membandingkan 2 nilai. Hasil dari operator ini adalah hasil perbandingan bernilai benar atau salah pada perhitungan matematika yang biasa digunakan. Pada matematika tanda < untuk mendeskripsikan sebuah nilai lebih kecil dibandingkan nilai lainnya, tanda > untuk mendeskripsikan sebuah nilai lebih besar dibandingkan nilai lainnya, dan tanda ≠ mendeskripsikan dua nilai yang berbeda (tidak sama dengan), dan tanda-tanda lainnya.

Pada bahasa pemrograman juga berlaku demikian, ketika akan membuat sebuah program sederhana dengan perbandingan nilai seperti pada matematika, hanya diperlukan menggunakan simbol atau operator pembanding yang dapat dimengerti oleh bahasa pemrograman. Simbol pembanding tersbut antara lain seperti berikut.

Tabel 3.4 Simbol-simbol Operator Pembanding

Simbol Matematika	Operator Pembanding
= (sama dengan)	=
≠ (tidak sama dengan)	< >
< (kurang dari)	<
> (lebih dari)	>
≤ (kurang dari atau sama dengan)	<=

≥ (lebih besar dari atau sama dengan)	>=

4. Operator Penaikan (*Increament*) dan Penurunan (*Decreament*)

Operator ini digunakan untuk menaikkan atau menurunkan suatu nilai integer (bilangan bulat) sebanyak satu (1) satuan, dan hanya dapat digunakan pada variabel. Operator-operator tersebut disimbolkan dengan simbol seperti berikut.

Tabel 3.5 Simbol *increament* dan *decreament*

Operator	Penjelasan
++	Penambahan dengan nilai 1
--	Pengurangan dengan nilai 1

Sebagai contoh seperti kode berikut.

```
x = x+1;
y = y-1;
```

Dapat ditulis menjadi:

```
x++;          ++x;
y--;          --y;
```

atau

5. Operator Penugasan (*Assignment*)

Operator penugasan merupakan operator yang paling sederhana dan sering digunakan dalam bahasa pemrograman pada sebuah

variabel. Operator tersebut digunakan dengan simbol atau tanda = (sama dengan). Misal, **String nama = Asnawi;**, **int angka = 7;**

Keunikan operator ini dalam Java adalah dapat menggunakannya secara berantai dalam suatu ekspresi, misal seperti berikut ini.

```
int a, b, c;
a = b = c = 30;
```

Contoh kode tersebut mengatur nilai 30 ke variabel a, b, dan c hanya menggunakan satu baris perintah. Hal ini dikarenakan pernyataan a = b = c = 30 akan dievaluasi oleh Java dari kanan ke kiri. Dimulai dari variabel c yang diatur dengan nilai 30, selanjutnya nilai variabel b diatur dengan nilai dari variabel c , dan nilai variabel a diatur dengan nilai dari variabel b, sehingga a, b, dan c bernilai 30. Pernyataan tersebut identik dengan kode berikut.

```
int a, b, c;
c = 30;
b = c;
a = b;
```

3.2 Struktur Dasar Pemrograman Java

Pemrograman Java memiliki struktur dasar sebagai berikut:

1. Komentar

Merupakan baris program yang tidak diproses sebagai perintah oleh kompiler dan interpreter. Baris komentar hanya berfungsi sebagai tanda keterangan tentang baris atau blok (kumpulan) perintah di bawahnya. Contoh penulisannya adalah:

```
//latihan java
```

Komentar tersebut menggunakan tanda // sebagai penanda komentar satu baris yang tidak akan diproses. Jika memberikan komentar yang panjang atau lebih dari satu baris dapat menggunakan tanda pembuka /* dan diakhiri dengan tanda */. Contohnya seperti berikut.

```
/* baris komentar ke-1
baris komentar ke-2... */
```

Selain tanda-tanda di atas, pada Java juga dapat memberikan komentar yang akan dianggap sebagai *javadoc comments* yang menggunakan tanda /** dan diakhiri dengan tanda */ . Komentar ini digunakan untuk memberikan dokumentasi tentang *class, data,* dan *method* yang digunakan.

2. Kata Kunci

   Merupakan kata-kata yang sudah memiliki arti khusus bagi interpreter dan kompiler Java untuk diterjemahkan menjadi perintah pada komputer untuk mengerjakan sesuatu. Kata kunci sebaiknya tidak digunakan sebagai nama variabel, konstanta, atau yang lainnya karena akan mengakibatkan kesalahan disaat proses kompilasi. Misal kompiler menemukan kata `class`, maka otomatis kompiler akan menerjemahkan bahwa itu adalah awal dari class dan nama class-nya ada di sebelah kanannya. Contoh lain dari kata kunci adalah `public`, `static`, dan `void`.

3. Modifier

   Java menggunakan beberapa kata khusus tertentu yang disebut *modifier*, yang berfungsi untuk menetapkan properti data, *method*, atau *class* dan bagaimana cara menggunakannya. Contoh dari kata khusus yang berupa *modifier* adalah `public`, `private`, `static`, `final`, `abstrac`, `protected`. Sebuah variabel atau class atau method yang memiliki modifier

public berarti dapat diakses oleh class lainnya, sedangkan bila modifiernya private maka tidak dapat diakses oleh class lainnya.

4. Statement

   Statement mempresentasikan sebuah aksi atau sebuah urutan aksi. Contohnya seperti berikut:

```
System.out.println("Hallo, Selamat Datang di Java!");
```

   Dari contoh tersebut akan menghasilkan kalimat yang terdapat dalam tanda kurung dan diantara tanda petik.

5. Blok

   Tanda *brace* (kurung kurawal) di dalam program yang membentuk sekelompok (1 blok) perintah atau komponen lain dalam program digunakan untuk membentuk sebuah struktur pada program seperti *class* atau *method*. Contohnya seperti berikut:

```
public class hallo {   //awal blok class
        public static void main(String[], args) {    //awal blok method
                System.out.println("Hallo, Selamat Datang di Java!");
        }    //akhir blok method
}    //akhir blok class
```

Setiap program Java setidaknya harus memiliki sebuah *class*, karena pada *Java class* adalah struktur program yang paling mendasar. Untuk melakukan pemrograman dengan menggunakan bahasa pemrograman Java, harus mengerti dasar-dasar pemrograman berorientasi objek dan mampu membuat *class*, dan menggunakannya di dalam pemrograman. Pada contoh program di bagian blok di atas, sudah terbentuk sebuah *class* yaitu *class hallo*.

6. Method

   Sebuah *class* dalam Java harus memiliki setidaknya sebuah *method*. Class utama harus memiliki *method* utama, jadi *method* harus ada di dalam *class* dan tidak dapat berdiri sendiri seperti sebuah fungsi di luar *class*.

7. Main Method

   Main method berfungsi mengontrol seluruh alur dari program ketika menjalankan tugasnya. Contohnya adalah seperti berikut.

```
public static void main(String[], args) {    //awal blok method
    System.out.println("Hallo, Selamat Datang di Java!");
}   //akhir blok method
```

BAB 4
STRUKTUR PERCABANGAN (*DECISION*)

Struktur percabangan pada Java ada beberapa perintah yaitu perintah ***IF, IF...ELSE, NESTED IF*** (IF bersarang/bertingkat), dan ***SWITCH...CASE***. Statemen IF digunakan untuk membuat percabangan dari struktur program Java. Beberapa struktur program dapat dibuat dan dijalankan salah satunya, atau beberapa diantaranya, jika ada kondisi yang terpenuhi. Struktur IF ini dikenali dan digunakan pada seluruh bahasa pemrograman dan secara umum untuk penggunaannya tidak jauh berbeda. Masing-masing struktur percabangan tersebut dapat dijelaskan seperti penjelasan di bawah ini.

4.1 IF

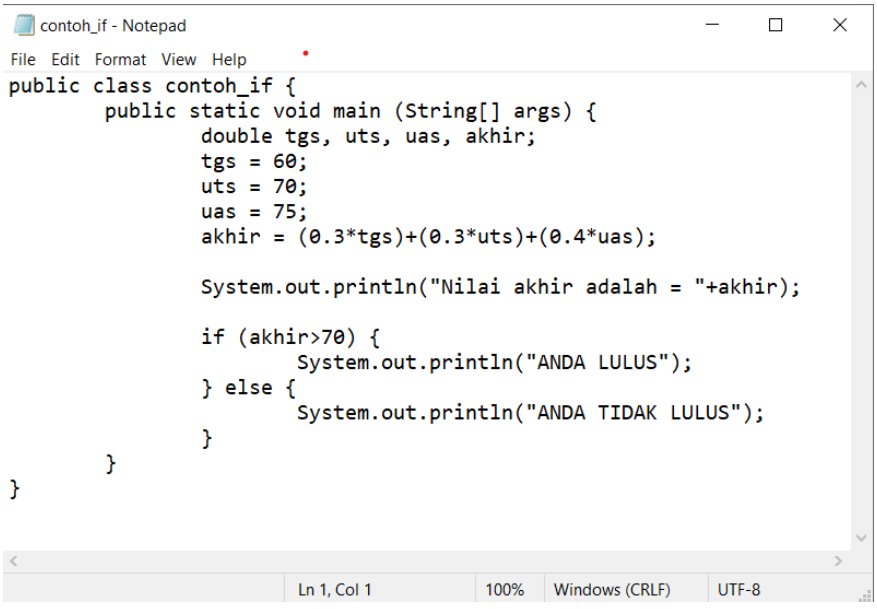
Kondisi IF dalam Java terdiri dari sebuah ekspresi boolean yang diikuti dengan satu statemen atau lebih. Kondisi ini digunakan untuk menyeleksi atau memeriksa apakah sebuah kondisi terpenuhi. Jika terpenuhi maka program akan melakukan perintah yang ada di bawahnya, dan jika tidak terpenuhi maka program akan melakukan perintah yang berikutnya, yang ada di luar IF. Berikut ini bentuk umum kondisi IF seperti pada gambar 4.1.

```
if (ekspresi_boolean) {
    //statemen yang akan dieksekusi jika
    //ekspresi_boolean bernilai benar
    statemen;
}
```

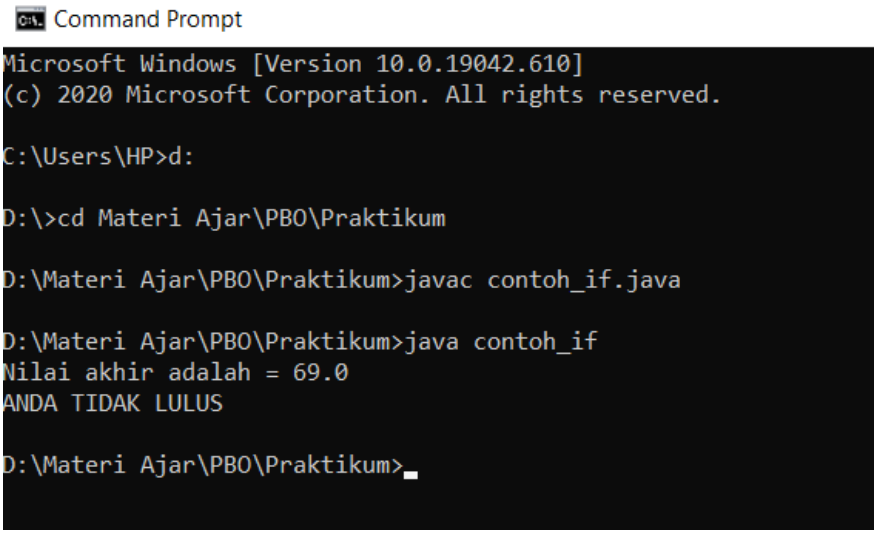
```
if (ekspresi_boolean) {
    //statemen yang akan dieksekusi jika
    //ekspresi_boolean bernilai benar
    statemen_1;
    statemen_2;
    statemen_n;
}
```

Gambar 4.1 Bentuk umum IF

Contoh program IF:



Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.



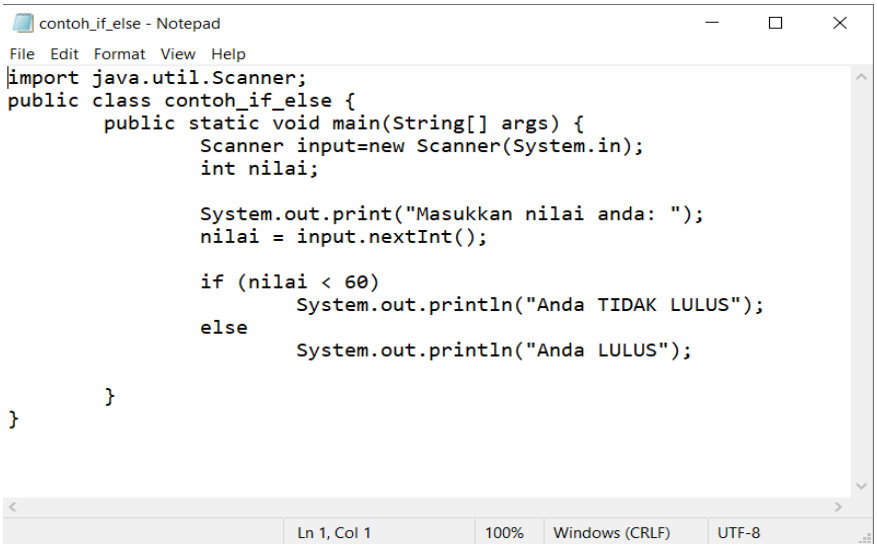
4.2 IF…ELSE

Bentuk lain dari kondisi IF adalah kondisi IF...ELSE. Bentuk ini merupakan bentuk yang banyak digunakan karena adanya statement yang akan dikerjakan jika kondisi yang diperiksa menghasilkan nilai *true* dan statement yang akan dikerjakan jika kondisi memberikan nilai *false*. Berikut ini bentuk kondisi IF...ELSE seperti pada gambar 4.2.

```
if (ekspresi_boolean) {
   //statement bernilai benar
   statement_1;
   statement_2;
   statement_n;
} else {
   //statement bernilai salah
   statement;
}
```

Gambar 4.2 Bentuk umum IF…ELSE

Contoh program:



Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

```
Command Prompt
Microsoft Windows [Version 10.0.19042.610]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd Materi Ajar\PBO\Praktikum

D:\Materi Ajar\PBO\Praktikum>java contoh_if_else
Masukkan nilai anda: 80
Anda LULUS

D:\Materi Ajar\PBO\Praktikum>java contoh_if_else
Masukkan nilai anda: 60
Anda LULUS

D:\Materi Ajar\PBO\Praktikum>java contoh_if_else
Masukkan nilai anda: 40
Anda TIDAK LULUS
```

4.3 NESTED IF (IF Bertingkat/Bersarang)

Merupakan struktur IF yang kompleks, terdiri dari beberapa struktur IF yang lain. Konsep dari NESTED IF adalah terdapat struktur IF yang ada di dalam struktur if lainnya. Berikut ini salah satu bentuk kondisi NESTED IF seperti pada gambar 4.3.

```
if (kondisi_A) {
     statement_A;
     if (kondisi_A1) {
          statement_A1;
     }
} else if (kondisi_B) {
     statement_B;
     if (kodisi_B1) {
          statement_B1;
     }
} else if (kondisi_n) {
     statement_n;
} else {
     statement_false;
}
```

Gambar 4.3 Salah satu bentuk umum NESTED IF

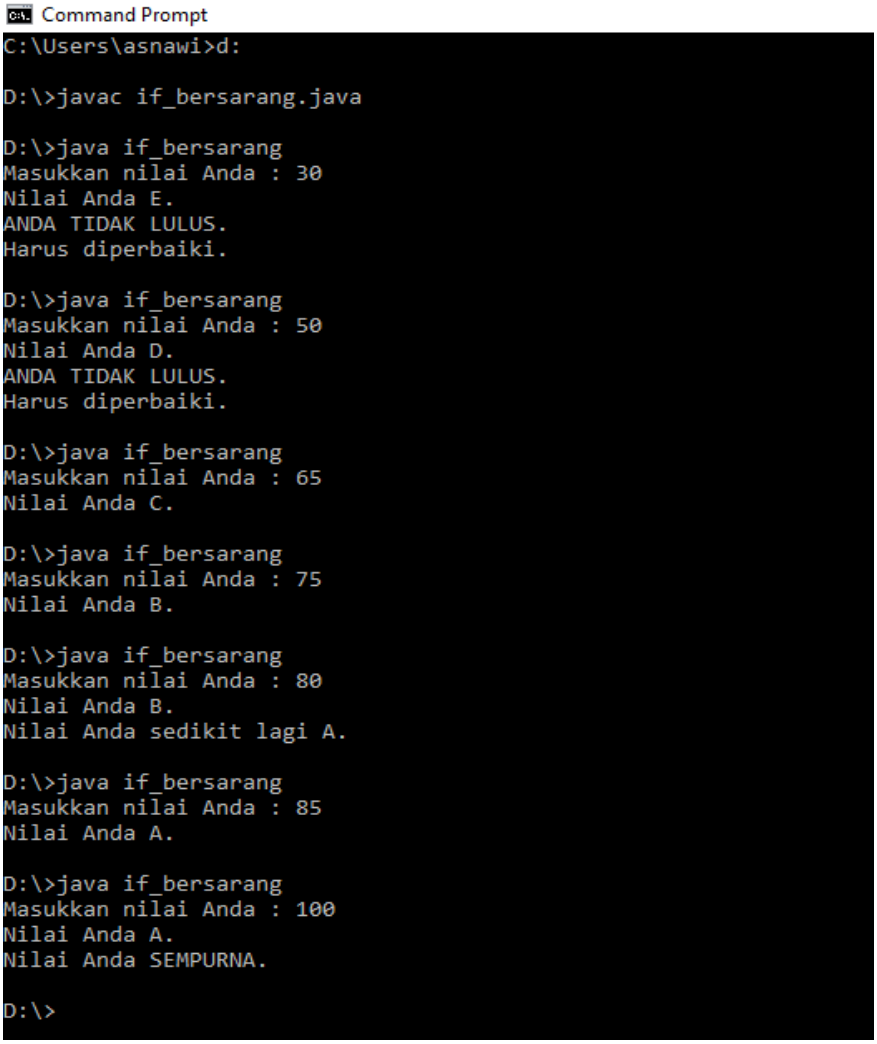
Contoh program:

if_bersarang - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class if_bersarang {
        public static void main(String[] args) {
                float nilai;
                Scanner input = new Scanner(System.in);

                System.out.print("Masukkan nilai Anda : ");
                nilai = input.nextFloat();

                if (nilai>80) {
                    System.out.println("Nilai Anda A.");
                    if (nilai==100) {
                        System.out.println("Nilai Anda SEMPURNA.");
                    }
                } else if (nilai>70) {
                    System.out.println("Nilai Anda B.");
                    if (nilai>=79) {
                        System.out.println("Nilai Anda sedikit lagi A.");
                    }
                } else if (nilai>60) {
                    System.out.println("Nilai Anda C.");
                } else if (nilai>40) {
                    System.out.println("Nilai Anda D.");
                    System.out.println("ANDA TIDAK LULUS.");
                    System.out.println("Harus diperbaiki.");
                } else {
                    System.out.println("Nilai Anda E.");
                    System.out.println("ANDA TIDAK LULUS.");
                    System.out.println("Harus diperbaiki.");
                }
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.


Command Prompt
C:\Users\asnawi>d:
D:\>javac if_bersarang.java
D:\>java if_bersarang
Masukkan nilai Anda : 30
Nilai Anda E.
ANDA TIDAK LULUS.
Harus diperbaiki.
D:\>java if_bersarang
Masukkan nilai Anda : 50
Nilai Anda D.
ANDA TIDAK LULUS.
Harus diperbaiki.
D:\>java if_bersarang
Masukkan nilai Anda : 65
Nilai Anda C.
D:\>java if_bersarang
Masukkan nilai Anda : 75
Nilai Anda B.
D:\>java if_bersarang
Masukkan nilai Anda : 80
Nilai Anda B.
Nilai Anda sedikit lagi A.
D:\>java if_bersarang
Masukkan nilai Anda : 85
Nilai Anda A.
D:\>java if_bersarang
Masukkan nilai Anda : 100
Nilai Anda A.
Nilai Anda SEMPURNA.
D:\>

4.4 SWITCH…CASE

Kondisi percabangan SWITCH…CASE merupakan kondisi percabangan alternatif dari kondisi percabangan IF. Kondisi ini menyederhanakan penulisan kondisi percabangan IF yang banyak. Selain itu kondisi SWITCH…CASE digunakan untuk memberikan kondisi dengan beberapa syarat yang identik, yang masing-masing mempunyai pernyataan yang berbeda-beda. Pada Java, nilai yang dilewatkan pada percangan SWITCH…CASE adalah nilai yang bertipe *int, short, byte/char*.

Jika kondisi memenuhi nilai 1, maka akan menjalankan pernyataan 1, dan break berfungsi untuk keluar dari SWITCH. Jika kondisinya memenuhi nilai 2, maka akan menjalankan pernyataan 2. Begitu seterusnya sesuai dengan banyak kondisi nilainya. Berikut ini merupakan bentuk umum dari struktur kondisi SWITCH…CASE seperti pada gambar 4.4.

```
switch (kondisi) {
     case nilai_1:
          pernyataan_1;
     break;
     case nilai_2:
          pernyataan_2;
     break;
     case nilai_3:
          pernyataan_3;
     break;
     case pernyataan_n:
          pernyataan_n;
     break;
     default:
          pernyataan_false;
}
```

Gambar 4.4 Bentuk umum SWITCH…CASE

Contoh program:

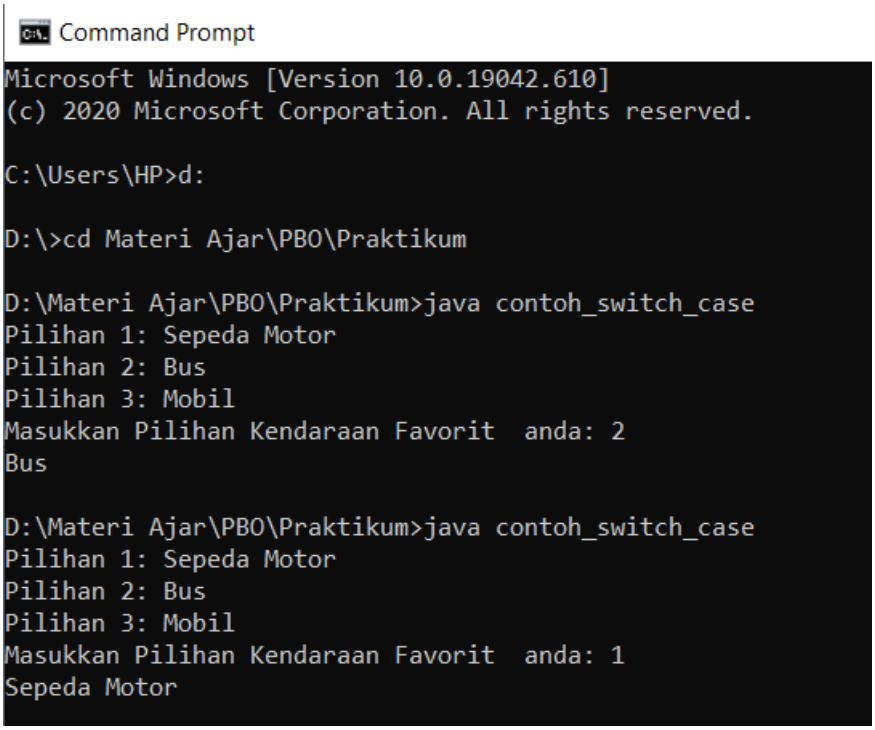
contoh_switch_case - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class contoh_switch_case {
        public static void main(String[] args) {
                System.out.println("Pilihan 1: Sepeda Motor");
                System.out.println("Pilihan 2: Bus");
                System.out.println("Pilihan 3: Mobil");

                Scanner input=new Scanner(System.in);
                System.out.print("Masukkan Pilihan Kendaraan Favorit  anda: ");
                int pilihan = input.nextInt();

                switch (pilihan) {
                case 1:System.out.println("Sepeda Motor");
                break;
                case 2:System.out.println("Bus");
                break;
                case 3:System.out.println("Mobil");
                break;
                default:System.out.println("Pilihan anda Tidak ada");
                }
        }
}
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.



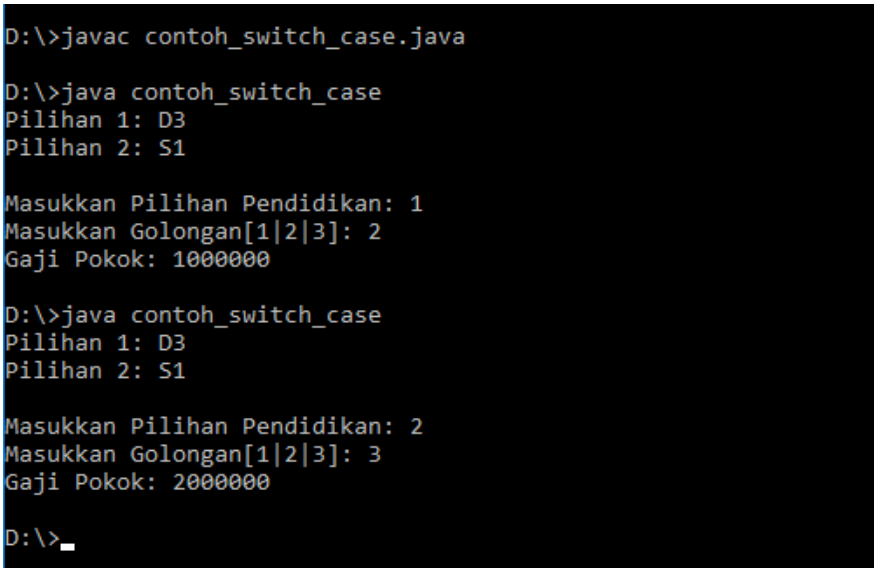
Selain contoh di atas, penggunaan SWITCH…CASE dapat dikombinasikan dengan kondisi IF…ELSE seperti contoh berikut.

contoh_switch_case - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class contoh_switch_case {
        public static void main(String[] args) {
                int pend, gol, gapok;
                System.out.println("Pilihan 1: D3");
                System.out.println("Pilihan 2: S1");
                System.out.println("");
                Scanner input = new Scanner(System.in);
                System.out.print("Masukkan Pilihan Pendidikan: ");
                pend = input.nextInt();

                System.out.print("Masukkan Golongan[1|2|3]: ");
                gol = input.nextInt();

                switch (pend) {
                case 1:
                if (gol == 1) {
                        gapok = 750000;
                        System.out.println("Gaji Pokok: "+gapok);
                } else if (gol == 2) {
                        gapok = 1000000;
                        System.out.println("Gaji Pokok: "+gapok);
                } else {
                        gapok = 1500000;
                        System.out.println("Gaji Pokok: "+gapok);
                }
                break;


               case 2:
               if (gol == 1) {
                       gapok = 1000000;
                       System.out.println("Gaji Pokok: "+gapok);
               } else if (gol == 2) {
                       gapok = 1500000;
                       System.out.println("Gaji Pokok: "+gapok);
               } else {
                       gapok = 2000000;
                       System.out.println("Gaji Pokok: "+gapok);
               }
               break;

               default:System.out.println("Pilihan Pendidikan TIDAK ADA");
               }
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.


D:\>javac contoh_switch_case.java
D:\>java contoh_switch_case
Pilihan 1: D3
Pilihan 2: S1
Masukkan Pilihan Pendidikan: 1
Masukkan Golongan[1|2|3]: 2
Gaji Pokok: 1000000
D:\>java contoh_switch_case
Pilihan 1: D3
Pilihan 2: S1
Masukkan Pilihan Pendidikan: 2
Masukkan Golongan[1|2|3]: 3
Gaji Pokok: 2000000
D:\>

BAB 5
STRUKTUR PERULANGAN (*LOOPING*)

Looping (perulangan) dalam bahasa pemrograman merupakan fitur yang memfasilitasi pelaksanaan pengaturan instruksi / fungsi berulang-ulang, sementara kondisi mengevaluasi ke nilai *true*. Ada situasi dimana diperlukan untuk menjalankan blok kode program beberapa kali. Secara umum pernyataan dieksekusi secara berurutan, pernyataan pertama dalam fungsi dijalankan pertama, diikuti yang kedua, dan seterusnya. Bahasa pemrograman Java menyediakan beberapa tipe *loop* untuk menangani persyaratan perulangan. Java menyediakan tiga cara untuk mengeksekusi *loop*. Meskipun semua cara menyediakan fungsi dasar yang serupa, ketiga cara tersebut berbeda dalam waktu pemeriksaan sintaks dan kondisi.

Struktur perulangan pada Java ada beberapa perintah yaitu perintah ***FOR***, ***WHILE***, dan ***DO...WHILE***. Statemen *FOR* digunakan untuk membuat perulangan dari struktur program Java. Beberapa struktur program dapat dibuat dan dijalankan secara berulang-ulang dengan batasan tertentu. Struktur FOR ini dikenali dan digunakan pada seluruh bahasa pemrograman dan secara umum untuk penggunaannya tidak jauh berbeda. Masing-

masing struktur perulangan tersebut dapat dijelaskan seperti penjelasan di bawah ini.

5.1 FOR

Perulangan *FOR* menyediakan cara sederhana penulisan struktur *loop*. Tidak seperti perulangan *WHILE*, pernyataan untuk inisialisasi kondisi, kondisi pengujian dan kenaikan/pengurangan dalam satu baris sehingga menyediakan struktur perulangan yang lebih singkat dan mudah untuk melakukan *debugging*. Pada gambar 5.1 berikut merupakan bentuk umum dari struktur FOR.

```
for (inisialisasi_kondisi; kondisi_pengujian; kenaikan/penurunan) {
    statement;
}
```

Gambar 5.1 Bentuk umum FOR

Proses perulangan FOR dapat dijelaskan seperti berikut ini:

1. *Initialization condition*

   Penginisialisasian variabel yang digunakan, pada tahap ini menandai dimulainya proses perulangan *FOR*. Variabel yang sudah dideklarasikan dapat digunakan.

2. *Testing condition*

Digunakan untuk menguji kondisi dalam satu perulangan, dan harus mengembalikan sebuah nilai *boolean*. Selain itu pengujian kondisi juga merupakan *entry control loop* (kontrol masuk perulangan), dimana kondisi diperiksa sebelum pernyataan perulangan dilaksanakan.

3. *Statement execution*

   Setelah kondisi dievaluasi ke nilai *true*, maka pernyataan dalam *loop body* (badan perulangan) dieksekusi.

4. *Increament/decreament*

   Digunakan untuk memperbarui variabel untuk iterasi selanjutnya.

5. *Loop termination*

   Ketika kondisi bernilai salah, perulangan berakhir yang menandai akhir dari sebuah perulangan dari siklusnya.

Contoh program:

contoh_for - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class contoh_for {
        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.print("Inputkan Batas Jumlah Bilangan: ");
                int batas = input.nextInt();

                System.out.print("Deret Bilangan Genap adalah:");
                for(int angka=2; angka<=batas; angka+=2)
                System.out.print(angka + " ");
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

Command Prompt

```
Microsoft Windows [Version 10.0.19042.610]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd Materi Ajar\PBO\Praktikum

D:\Materi Ajar\PBO\Praktikum>javac contoh_for.java

D:\Materi Ajar\PBO\Praktikum>java contoh_for
Inputkan Batas Jumlah Bilangan: 10
Deret Bilangan Genap adalah:2 4 6 8 10
```

Perulangan *FOR* juga memiliki bentuk lain yang dirancang untuk iterasi melalui koleksi atau *array*. Bentuk ini kadang-kadang disebut sebagai pernyataan yang disempurnakan, dan dapat membuat perulangan menjadi lebih ringkas dan mudah dibaca. Perulangan tersebut yaitu *Enhanced For*.

Enhanced For Loop

Java juga menyertakan versi lain untuk perulangan yang diperkenalkan di Java 5. *Enhanced For Loop* menyediakan cara yang lebih sederhana untuk melakukan iterasi melalui elemen koleksi atau *array*. Namun perulangan model ini hanya digunakan ketika ada kebutuhan untuk melakukan iterasi melalui elemen secara berurutan tanpa mengetahui indeks dari elemen yang diproses saat ini. Gambar 5.2 berikut merupakan bentuk umum dari struktur *Enhanced For*.

```
for (T elemen:koleksi array) {
    statement;
}
```

Gambar 5.2 Bentuk umum *Enhanced For*

Contoh program:

contoh_enhancedfor - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class contoh_enhancedfor {
        public static void main(String[] args) {
                String array[] = {"Andi", "Budi", "Dani"};

                for (String x:array) {
                    System.out.println(x);
                }
        }
}
```

Atau dalam bentuk FOR seperti berikut.

contoh_for_3 - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class contoh_for_3 {
        public static void main(String[] args) {
                String array[] = {"Andi", "Budi", "Dani"};

                for (int x=0; x<array.length; x++) {
                    System.out.println(array[x]);
                }
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

Enhanced For:

Command Prompt

```
Microsoft Windows [Version 10.0.19042.610]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd Materi Ajar\PBO\Praktikumk Java
The system cannot find the path specified.

D:\>cd Materi Ajar\PBO\Praktikum Java

D:\Materi Ajar\PBO\Praktikum Java>javac contoh_enhancedfor.java

D:\Materi Ajar\PBO\Praktikum Java>java contoh_enhancedfor
Andi
Budi
Dani
```

For:

```
D:\Materi Ajar\PBO\Praktikum Java>javac contoh_for_3.java

D:\Materi Ajar\PBO\Praktikum Java>java contoh_for_3
Andi
Budi
Dani
```

5.2 WHILE

WHILE LOOP (perulangan *while*) merupakan pernyataan aliran kontrol yang memungkinkan kode dieksekusi berulang kali berdasarkan kondisi boolean yang diberikan. Perulangan *while* dapat dianggap sebagai pernyataan yang berulang. Gambar 5.3 berikut merupakan bentuk umum dari struktur *WHILE*.

```
while (kondisi boolean) {
    statement;
}
```

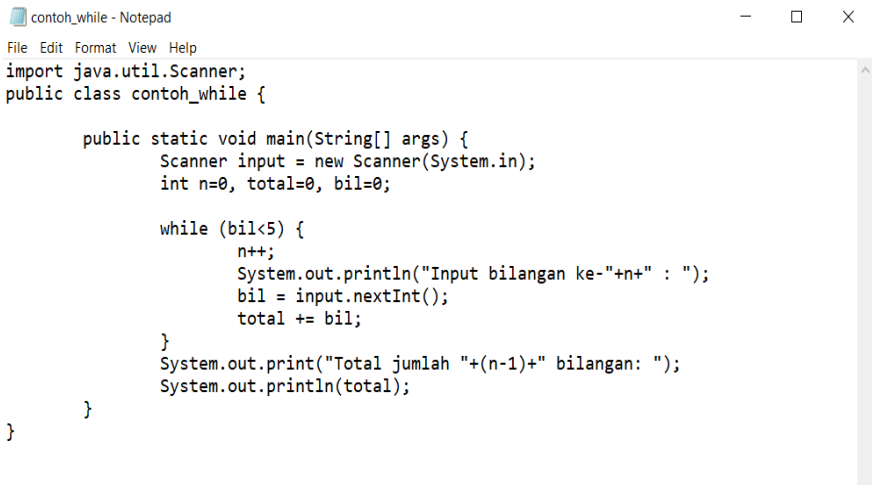
Gambar 5.3 Bentuk umum *WHILE*

Proses perulangan WHILE dapat dijelaskan seperti berikut ini:

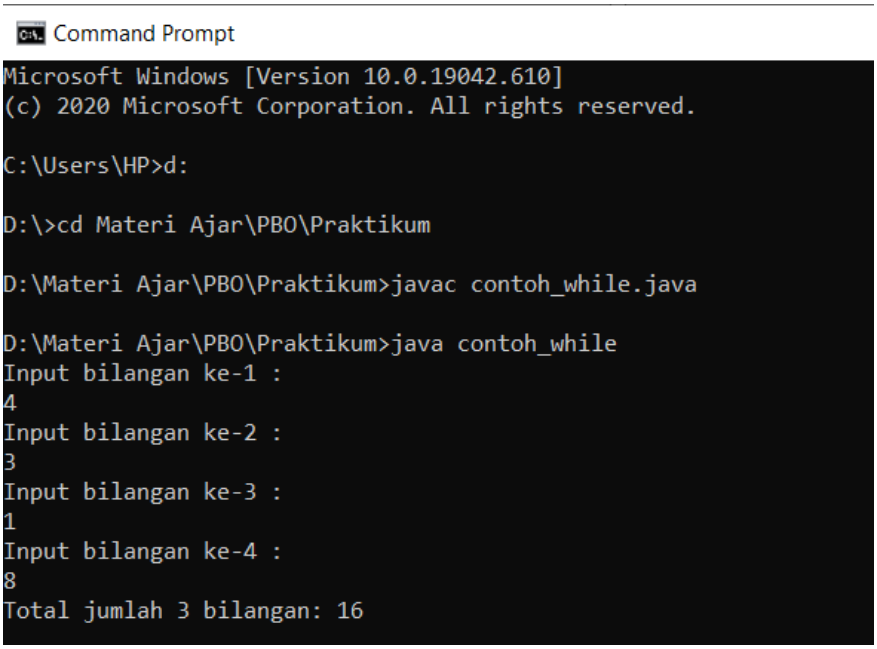
1. Perulangan *WHILE* dimulai dengan pengecekan kondisi. Jika dievaluasi benar, maka pernyataan dalam badan perulangan (*body loop*) dieksekusi. Hal ini disebut juga sebagai *entry control loop* (kontrol masuk perulangan).

2. Setelah kondisi dievaluasi ke nilai yang benar (*true*), pernyataan dalam badan perulangan dieksekusi. Biasanya pernyataan berisi nilai pembaruan untuk variabel yang sedang diproses untuk iterasi selajutnya.
3. Ketika kondisi menjadi salah (*false*), maka perulangan berakhir yang menandai akhir dari siklusnya.

Contoh program:



Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.



5.3 DO…WHILE

Perulangan *DO...WHILE* hampir sama dengan perulangan *WHILE*, hanya berbeda pada pemeriksaan kondisi dimana dilakukan setelah mengeksekusi pernyataan. Hal ini disebut juga sebagai *exit control loop* (kontrol keluar perulangan). Atau bisa dikatakan perulangan *DO...WHILE* proses evaluasi pernyataannya dibagian bawah *loop*. Gambar 5.4 berikut merupakan bentuk umum dari struktur *DO...WHILE*.

Gambar 5.4 Bentuk umum DO…WHILE

Proses perulangan DO...WHILE dapat dijelaskan seperti berikut ini:

1. Perulangan DO...WHILE dimulai dengan eksekusi pernyataan. Tidak ada pemeriksaan apapun untuk pertama kali dilaksanakan.
2. Setelah eksekusi pernyataan dan pembaruan nilai variabel, kondisi diperiksa untuk nilai benar (*true*) atau salah (*false*). Jika dievaluasi ke nilai *true*, maka iterasi perulangan berikutnya dimulai.
3. Ketika kondisi menjadi bernilai *false*, perulangan berakhir yang menandai akhir dari siklusnya.
4. Penting untuk dicatat bahwa perulangan DO...WHILE akan mengeksekusi pernyataannya minimal satu kali sebelum kondisi apapun diperiksa, dan oleh karena itu hal ini juga bisa disebut sebagai *exit control loop* (kontrol keluar perulangan).

Contoh program:

contoh_dowhile - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class contoh_dowhile {
        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                int n=0, total=0, bil=0;

                do {
                        n++;
                        System.out.println("Input bilangan ke-"+n+" : ");
                        bil = input.nextInt();
                        total += bil;

                        System.out.print("Total jumlah "+(n-1)+" bilangan: ");
                        System.out.println(total);
                }
                while (bil<5);
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

```
D:\Materi Ajar\PBO\Praktikum>javac contoh_dowhile.java

D:\Materi Ajar\PBO\Praktikum>java contoh_dowhile
Input bilangan ke-1 :
3
Total jumlah 0 bilangan: 3
Input bilangan ke-2 :
4
Total jumlah 1 bilangan: 7
Input bilangan ke-3 :
2
Total jumlah 2 bilangan: 9
Input bilangan ke-4 :
7
Total jumlah 3 bilangan: 16
```

BAB 6
STRUKTUR ARRAY

Ada saat dimana dibutuhkan untuk menyimpan banyak nilai selama melakukan eksekusi suatu program. Misalnya diperlukan untuk membaca 100 angka, menghitung rata-ratanya, dan ingin mengetahui berapa banyak yang nilainya di atas rata-rata. Pertama program akan membaca angka-angka yang dimasukkan dan menghitung rata-ratanya, kemudian membandingkan setiap angka dengan rata-rata untuk menentukan apakah angka itu bernilai lebih dari nilai rata-rata.

Dalam penyelesaian pekerjaan ini, angka-angkat tersebut disimpan di dalam variabel-variabel. Harus dilakukan pendeklarasian 100 variabel untuk menampung angka-angka tersebut dan secara berulang-ulang menulis kode yang sama sebanyak 100 kali. Penulisan progra seperti ini tentu sangat tidak efisien dan melelahkan.

Pada pemrograman Java dan kebanyakan bahasa pemrograman tingkat tinggi yang lainnya telah menyediakan suatu struktur data yang disebut dengan array, yang mampu menyimpan koleksi elemen-elemen secara sekuensial dengan tipe data yang sama. Pada kasus di atas dapat dilakukan

penyimpanan 100 angka ke dalam suatu array dan mengaksesnya melalui suatu variabel array tunggal.

6.1 Dasar Array

Suatu array digunakan untuk menyimpan koleksi data, atau merupakan sebuah koleksi variabel bertipe data yang sama.

6.1.1 Pendeklarasian Variabel Array

Penggunaan array dalam suatu program, perlu mendeklarasikan suatu variabel untuk mereferensi array dan menspesifikasikan tipe elemen array. Sintaks untuk mendeklarasikan sembarang variabel array seperti ini

```
tipeElemen[ ] varRefArray;
```

`tipeElemen` dapat berupa sembarang tipe data, dan semua elemen di dalam array harus memiliki tipe data yang sama. Sebagai contoh kode berikut ini mendeklarasikan suatu variabel `myData` yang mereferensi suatu array dengan elemen-elemen int.

```
int[ ] myData;
```

6.1.2 Membuat Array

Berbeda dengan pendeklarasian variabel tipe data primitif, deklarasi suatu variabel array tidak mengalokasikan memori untuk array tersebut.

Pendeklarasian hanya menyimpan lokasi untuk referensi kepada suatu array. Jika suatu variabel tidak memuat suatu referensi kepada array, maka nilai variabel tersebut adalah *null*. Setelah suatu variabel array dibuat, bisa dilakukan pembuatan array menggunakan operator new dengan sintaks seperti berikut ini:

```
varRefArray = new tipe Elemen[ukuranArray];
```

Statemen ini melakukan dua hal yaitu membuat suatu array menggunakan `new tipeElemen[ukuranArray]` dan menugaskan referensi array kepada `varRefArray`.

Pendeklarasian suatu variabel array, membuat array, dan menugaskan referensi array kepada variabel dpat digabungkan menjadi satu statemen seperti berikut ini.

```
tipeElemen varRefArray = new
tipeElemen[ukuranArray];
```

atau

```
tipeElemen varRefArray[ ] = new
tipeElemen[ukuranArray];
```

Contoh dari statemen sebelumnya:

```
int[ ] myData = new int[5];
```

Statemen tersebut mendeklarasikan suatu variabel array myData, membuat suatu array dengan 5 elemen *int*, dan menugaskan referensinya kepada `myData`.

6.1.3 Ukuran Array dan Nilai Default

Supaya memori untuk suatu array dialokasikan, ukuran array harus diberikan untuk menentukan jumlah elemen yang bisa disimpan di dalam suatu array. Ukuran suatu array tidak bisa diubah setelah array dibuat. Ukuran array bisa diperoleh menggunakan `varRefArray.length`.

Sebagai contoh:

`myData.length` adalah 5.

6.1.4 Variabel Berindeks

Elemen-elemen array diakses lewat indeks. Indeks array dimulai dari nol (0) hingga `varRefArray.length-1`. Setiap elemen di dalam array direpresentasikan menggunakan sintaks seperti berikut ini, yang dikenal sebagai variabel berindeks.

```
varRefArray[indeks];
```

Sebagai contoh, `varRefArray[4]` merepresentasikan elemen terakhir pada array `myData`. Setelah sebuah array dibuat, suatu indeks variabel bisa

digunakan seperti variabel biasa. Misalnya, kode berikut menambahkan nilai-nilai di dalam **myData[0]**, **myData[1]**, dan **myData[2]**:

```
myData[2] = myData[0] + myData[1]
```

Perulangan berikut menugaskan nol (0) kepada **myData[0]**, satu (1) kepada **myData[1]** hingga empat (4) kepada **myData[4]**.

```
for (int i=0; i<myData.length; i++) {
    myData[i] = i;
}
```

6.1.5 Penginisialisasi Array

Java mempunyai notasi *shorthand*, yang dikenal sebagai penginisialisasi array. Notasi ini menggabungkan deklarasi sebuah array, membuat sebuah array, dan menginisialisasi sebuah array dalam satu statemen, dengan menggunakan sintaks seperti berikut ini.

```
tipeElemen[ ] varRefArray = {nilai_0, nilai_1,
....., nilai_n};
```

Contoh:

```
int[ ] myData = {1, 2, 3, 4};
```

Statemen di atas mendeklarasikan, membuat, dan menginisialisasi array **myData** dengan empat (4) elemen, yang ekivalen dengan beberapa statemen berikut ini:

```
int[] myData = new int[4];
myData[0] = 1;
myData[1] = 2;
myData[2] = 3;
myData[3] = 4;
```

Perhatikan operator ***new*** tidak digunakan dalam sintaks penginisialisasian array. Jika menggunakan sintaks penginisialisasi array, maka harus mendeklarasikan, membuat, dan menginisialisasi array dalam satu (1) statemen seperti berikut ini.

```
int[] myData;
myData = {1, 2, 3, 4};
```

6.1.6 Memproses Array

Saat memproses elemen-elemen array, seringkali menggunakan tipe perulangan for, karena terdapat 2 alasan yaitu:

1. Semua elemen suatu array bertipe sama.
2. Ukuran array yang telah diketahui, maka cocok untuk menggunakan tipe perulangan *for*.

Berikut ini beberapa contoh pemrosesan array:

1. Perulangan ini menginisialisasi array **myData** dengan nilai-nilai dari *user* (pengguna), contohnya seperti berikut.

array_1 - Notepad

File Edit Format View Help

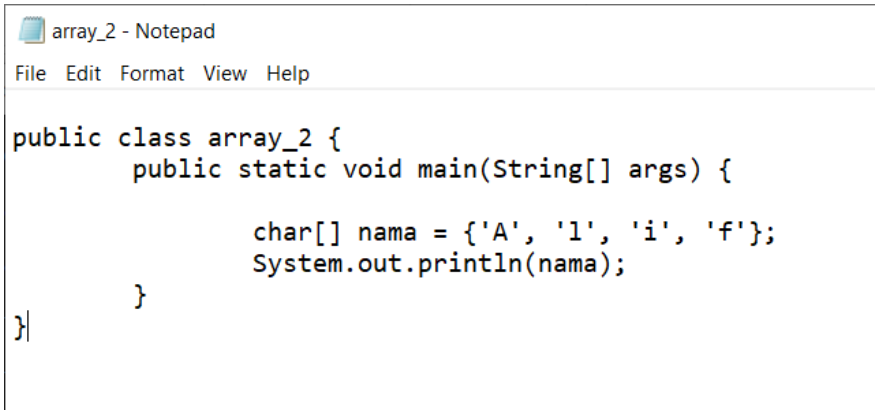
```
import java.util.Scanner;
public class array_1 {
        public static void main(String[] args) {
                int x, batas;
                Scanner input = new Scanner(System.in);
                System.out.print("Masukkan batas angka: ");
                batas = input.nextInt();
                int[] myData = new int[batas];

                for (x=0; x<myData.length; x++)
                System.out.print(x+" ");
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

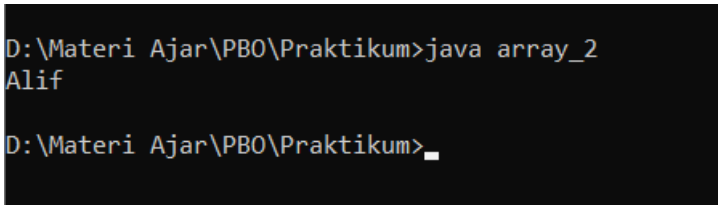
Command Prompt

```
Microsoft Windows [Version 10.0.19042.610]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd Materi Ajar\PBO\Praktikum

D:\Materi Ajar\PBO\Praktikum>javac array_1.java

D:\Materi Ajar\PBO\Praktikum>java array_1
Masukkan batas angka: 9
0 1 2 3 4 5 6 7 8
D:\Materi Ajar\PBO\Praktikum>_
```

2. Untuk array bertipe ***char***, bisa ditampilkan menggunakan satu statemen seperti contoh berikut.



Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.



3. Menggunakan suatu variabel untuk menyimpan penjumlahan elemen-elemen array. Inisialisasikan variabel dengan nol (0). Tambahkan setiap elemen di dalam array ke dalam variabel menggunakan perulangan. Contoh program:

array_3 - Notepad

File Edit Format View Help

```
import java.util.Scanner;
public class array_3 {
        public static void main(String[] args) {
                int x, batas, total=0;
                Scanner input = new Scanner(System.in);
                System.out.print("Masukkan batas angka: ");
                batas = input.nextInt();
                int[] myData = new int[batas];

                for (x=0; x<myData.length; x++)
                total += x;
                System.out.print("Jumlah total bilangan dalam array: "+total);
        }
}
```

Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

```
D:\Materi Ajar\PBO\Praktikum>javac array_3.java

D:\Materi Ajar\PBO\Praktikum>java array_3
Masukkan batas angka: 9
Jumlah total bilangan dalam array: 36
D:\Materi Ajar\PBO\Praktikum>
```

6.2 Penyalinan Array

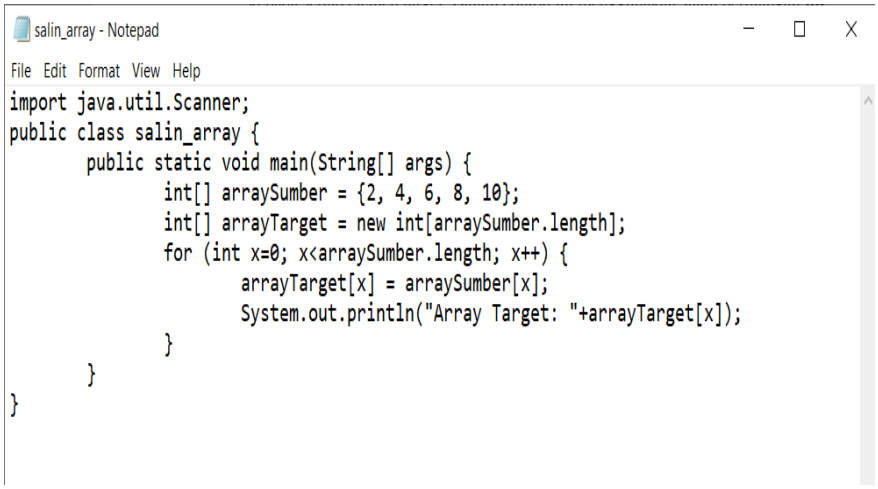
Di dalam sebuah program perlu melakukan penyalinan suatu array atau sebagian dari suatu array. Pada kasus seperti itu biasanya akan menggunakan operator penugasan seperti berikut:

```
data2 = data1;
```

Statemen ini tidak menyalin atau menduplikasi isi array yang direferensi oleh **data1** kepada **data2**, tetapi hanya menyalin nilai referensi **data1** kepada **data2**. Setelah statemen tersebut dieksekusi, **data1** dan **data2** mereferensi ke array yang sama. Dalam Java dapat menggunakan operator penugasan untuk menyalin variabel-variabel tipe data primitif, tetapi tidak untuk menyalin array. Menugaskan suatu variabel array kepada variabel array yang lain sesungguhnya hanya menyalin satu referensi kepada referensi yang lain dan membuat kedua referensi menunjuk ke lokasi memori yang sama.

Terdapat tiga (3) cara untuk menyalin array yaitu:

1. Gunakan suatu loop untuk menyalin elemen-elemen individual array satu per satu. Dapat menggunakan suatu perulangan untuk menyalin setiap elemen array sumber kepada setiap elemen target. Dalam contoh ini menggunakan suatu perulangan *for*.



Dari contoh tersebut jika dijalankan menggunakan Command Prompt akan menghasilkan seperti berikut ini.

```
D:\Materi Ajar\PBO\Praktikum>javac salin_array.java

D:\Materi Ajar\PBO\Praktikum>java salin_array
Array Target: 2
Array Target: 4
Array Target: 6
Array Target: 8
Array Target: 10
```

Dari hasil tersebut terlihat bahwa arrayTarget menyalin setiap elemen dari arraySumber dengan nilai 2, 4, 6, 8, dan 10.

2. Gunakan metode ***arraycopy*** dalam kelas System.
3. Gunakan metode ***clone*** untuk menyalin array.

BAB 7
METHOD DAN PARAMETER

7.1 Method pada Java

Method merupakan sekumpulan kode yang diberikan nama, untuk merujuk ke sekumpulan kode tersebut digunakan sebuah nama yang disebut sebagai nama method. Method memiliki parameter sebagai masukan (*input*) dan nilai kembalian sebagai keluaran (*output*).

Deklarasi method terdiri dari beberapa bagian yaitu antara lain:

1. Bagian access modifier dari method, yaitu *public, private, protected*, atau *default*.
2. Bagian tipe kembalian dari method, jika method tidak mengembalikan nilai apa-apa maka keyword void yang digunakan.
3. Bagian nama method, sesuai dengan aturan java code convention, nama method diawali dengan huruf kecil dan setiap kata setelahnya diawali dengan huruf besar (*camel case*).
4. Setelah nama method, terdapat parameter. Sebuah method bisa saja tidak memiliki parameter, memiliki satu (1), dua (2), dan seterusnya. Setelah Java 5 terdapat fitur yang disebut dengan

varargs, fitur ini memungkinkan method untuk memiliki jumlah parameter yang bervariasi.

5. Bagian terakhir dari method ialah deklarasi *throws exception*, dimana dapat dideklarasikan tipe exception yang akan di throws oleh method.

Nama method dan parameter ialah pembeda antara satu method dengan method yang lainnya. Jika ada dua method namanya berbeda, maka kedua method itu berbeda. Dan jika ada dua method yang namanya sama namun dianggap sebagai method yang berbeda jika parameternya berbeda. Method dengan nama yang sama dan parameternya berbeda ini dalam konsep *Object Oriented Programming* (OOP) disebut sebagai ***overloading***.

Jika method memiliki nama yang sama dan parameter yang sama tetapi tipe return atau throws exception-nya berbeda, maka akan menyebabkan error pada saat program dikompilasi. Jadi, jika mendefinisikan method baru pastikan namanya tidak sama dengan method yang lain, atau setidaknya parameternya berbeda baik itu dari sisi jumlahnya, tipe parameter atau posisi parameter.

Berikut ini contoh dari sebuah main method:

```
12  public class HelloJava {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          System.out.println("Hello Java...!!");
20      }
21
22  }
```

Pada contoh tersebut dapat dilihat bahwa main methodnya adalah:

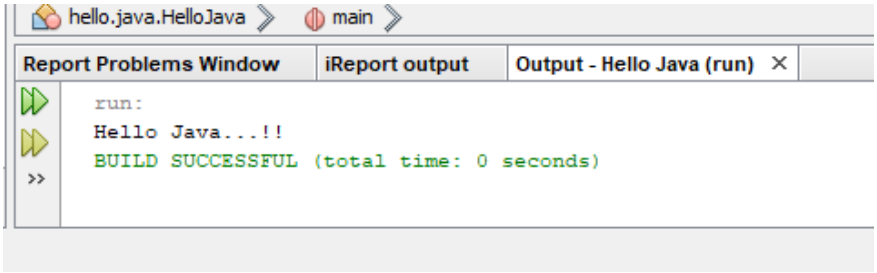
`public static void main(String[ ] args)`

Selanjutnya contoh untuk sebuah method:

```
11      */
12  public class HelloJava {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          System.out.println("Hello Java...!!");
20      }
21
22      public static void contohMethod() {
23          System.out.println("Hello Java..!!");
24      }
25
26  }
27
```

Pada contoh di atas, methodnya dengan nama **contohMethod()**.

Namun contoh di atas jika di compile dan dijalankan maka method baru tersebut, hasilnya tidak dapat ditampilkan atau dilihat pada jendela *output*.

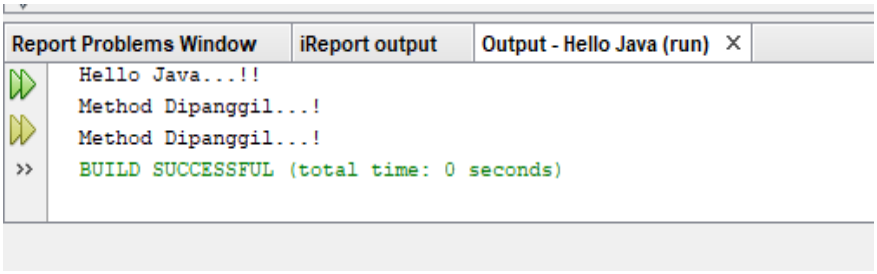


Terdapat beberapa kata kunci (*keyword*) yang dapat ditambahkan ke dalam deklarasi method. Kata kunci yang sering digunakan adalah static. Bagian kode Java yang dideklarasikan dengan menggunakan static akan menjadi anggota dari class, bukan anggota dari object. Karena method yang ditandai dengan statici adalah bagian dari class, maka dapat diakses langsung dari nama class itu sendiri, tanpa membuat object. Method static hanya dapat memanggil method lain dalam satu class yang juga diatandai dengan static. Method main yang biasa digunakan untuk menjalankan aplikasi Java juga ditandai dengan static, misal akan memanggil method lain dari method static, maka method yang lainnya juga harus ditandai dengan static. Berikut ini sebagai contohnya:

```java
public class HelloJava {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Hello Java...!!");
        //static method memanggil static method yang lain dalam satu class
        contohMethod();
        //method static juga dapat dipanggil dari nama classnya
        HelloJava.contohMethod();
    }

    public static void contohMethod() {
        System.out.println("Method Dipanggil...!");
    }

}
```

Terlihat bahwa di dalam main method di atas terdapat method yang dipanggil yaitu method **contohMethod()** dipanggil 2 kali, baik menggunakan nama class atau tidak.

Dari contoh tersebut di atas, jika di compile dan dijalankan akan menghasilkan keluaran seperti berikut.



Terlihat bahwa hasil keluarannya menampilkan kalimat yang sama sebanyak dua kali, sama seperti pemanggilan method **contohMethod()** sebanyak 2 kali.

7.2 Parameter pada Java

Dalam pemrograman komputer, parameter atau sebuah argumen formal adalah jenis variabel khusus yang digunakan dalam subroutine untuk merujuk kepada salah satu bagian data yang disediakan sebagai masukan ke subroutine. Daftar parameter yang terurut biasanya termasuk dalam definisi subroutin, sehingga setiap kali subroutin dipanggil, argumennya untuk panggilan tersebut dievaluasi dan nilai yang dihasilkan dapat ditetapkan ke parameter yang sesuai.

Argumen dalam ilmu komputer merupakan ekspresi masukan yang sebenarnya dilewatkan ke fungsi, prosedur, atau routine dalam pernyataan panggilan, sedangkan parameter merupakan variabel di dalam pelaksanaan subroutine. Sebagai contoh, jika mendefinisikan subroutin sebagai **def add(x, y): return x+y**, maka **x** dan **y** adalah parameter. Sedangkan add(2, 3) maka 2 dan 3 adalah argumen.

Perhatikan bahwa variabel dari konteks panggilan dapat berupa argumen, jika subroutin disebut sebagai a=2, b=3, add(a, b) adalah argumen, bukan yang nilai 2 dan 3. Program berikut ini mendefinisikan fungsi bernama “harga_jual” dan memiliki satu parameter bernama “harga” bertipe data double, tipe kembalian (function’s return) juga bertipe data double.

```
double harga_jual(double harga)
{
    return 0.5*harga;
}
```

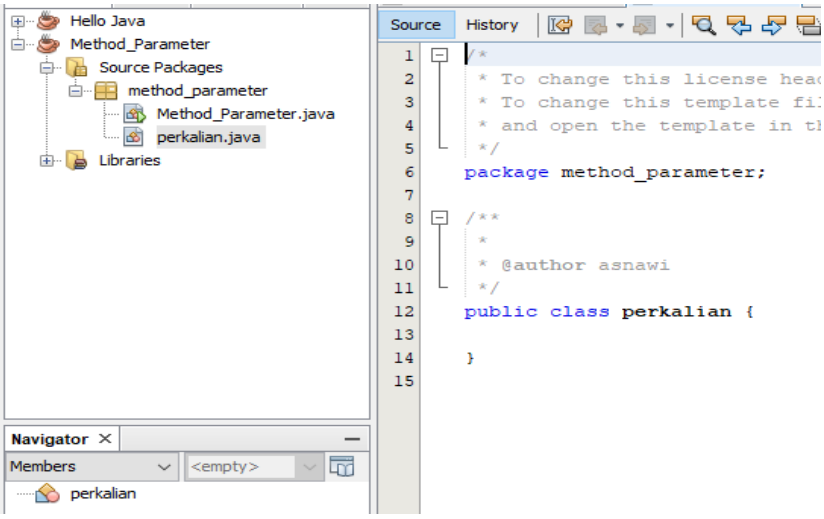
Setelah fungsi ditentukan,fungsi tersebut dapat dipanggil seperti berikut:

```
harga_jual(10);
```

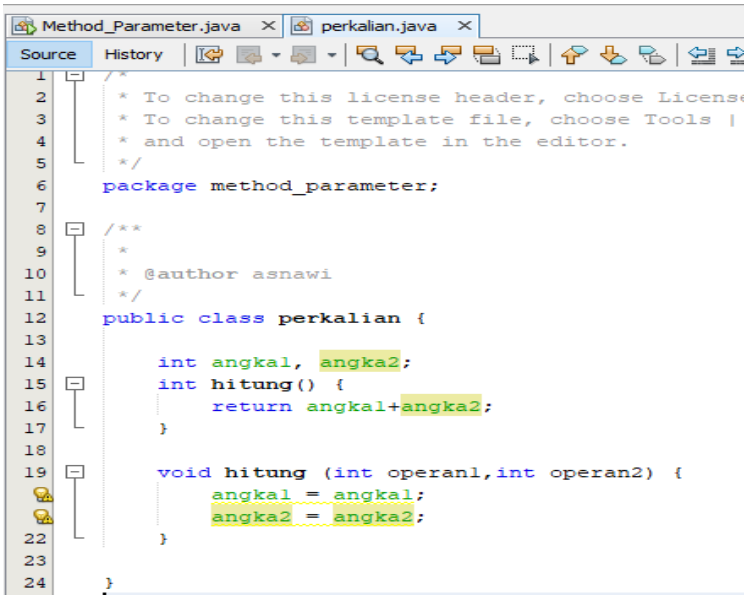
Pada contoh tersebut fungsi telah dipanggil dengan argumen 10 dan nilai tersebut akan ditetapkan kepada “harga”, dan fungsi mulai menghitung hasilnya. Langkah-langkah untuk menghasilkan sebuah hasil nilai terlampir dalam tanda kurung kurawal { }. Nilai 0.5*harga menunjukkan bahwa yang dilakukan pertama kali ialah mengalikan 0.5 dengan nilai “harga”, yang memberikan hasil nilai 5.

Contoh implementasi method dan parameter:

Buat aplikasi baru dengan nama Method_Parameter.



Buat kelas dengan nama **perkalian**, kemudian isikan kode program seperti berikut.



Kemudian isikan kode program pada main method **Method_Parameter** seperti berikut.

```java
/**
 * @author asnawi
 */
public class Method_Parameter {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        perkalian perkalian1 = new perkalian();
        int kali;

        perkalian1.angka1 = 12;
        perkalian1.angka2 = 5;
        System.out.println();

        kali = perkalian1.angka1*perkalian1.angka2;
        System.out.println("Contoh method & parameter pada perkalian..!");
        System.out.println("Perkalian 12*5 = "+kali);

    }

}
```

Hasil *running* program seperti berikut:

```
Contoh method & parameter pada perkalian..!
Perkalian 12*5 = 60
BUILD SUCCESSFUL (total time: 0 seconds)
```

BAB 8
IMPLEMENTASI PEMROGRAMAN BERBASIS OBJEK

8.1 Sifat Pemrograman Berbasis Objek

Pemrograman berorientasi objek merupakan sebuah paradigma pemrograman yang menekankan pada penciptaan objek yang di dalamnya terdapat atribut (berbentuk variabel) dan *method* (berbentuk prosedur atau fungsi). Di dalam pemrograman berorientasi objek terdapat 2 istilah yang sering dipakai yaitu objek dan kelas.

Kelas merupakan kumpulan objek-objek yang memiliki atribut dan *method* yang sama. Dari sebuah kelas bisa dibentuk banyak objek dengan nama yang berbeda. Objek merupakan hasil instansiasi dari sebuah kelas. Hubungan antara kelas dan objek dapat dicontohkan seperti tabel 8.1 berikut:

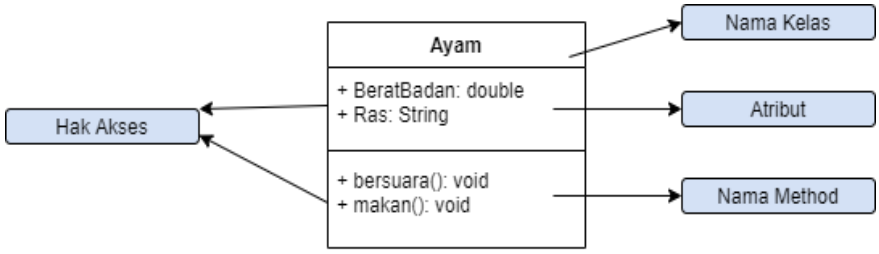
Tabel 8.1 Contoh Kelas dan Objek

Kelas	Objek
Ayam	Ayam Petelur
	Ayam Pedaging
	Ayam Kampung

Sifat yang dimiliki oleh pemrograman berorientasi objek yaitu:

1. Enkapsulasi (*Encapsulation*)

Merupakan sebuah proses penggabungan atribut dan *method* ke dalam sebuah kelas. Enkapsulasi juga dihubungkan dengan pemberian hak akses pada kelas ataupun anggota dari sebuah kelas (atribut dan *method*). Ilustrasi sederhana dari proses enkapsulasi seperti pada gambar 8.1 berikut.



Gambar 8.1 Ilustrasi Enkapsulasi

Pada gambar 8.1, atribut dan *method* dibungkus dalam 1 kelas yaitu **Ayam**. Enkapsulasi membuat atribut dan *method* menjadi lebih terorganisir dan jelas kepemilikannya. Atribut “BeratBadan” dan “Ras”, serta *method* “bersuara()” dan “makan()” adalah milik kelas ayam.

Selain itu pemberian hak akses juga merupakan bagian penting dari proses enkapsulasi. Hak akses diberikan pada 3 hal yaitu kelas, atribut dan *method*. Terdapat 3 jenis hak akses yang biasa digunakan dalam Java yaitu:

a. Public

Hak akses ini digunakan jika atribut atau *method* ingin bisa diakses oleh kelas lain (bukan kelas pemilik atribut dan *method* tersebut). Pada kelas diagram, hak akses ini diwakili dengan simbol plus (+).

b. Private

Hak akses ini digunakan jika atribut atau *method* hanya ingin diakses pada kelas pemilik. Pada kelas diagram, hak akses ini diwaliki dengan simbol minus (-).

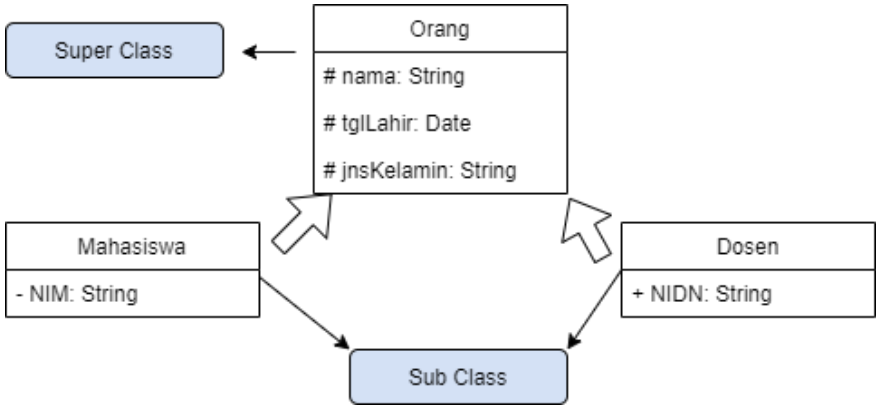
c. Protected

Hak akses ini digunakan pada konsep pewarisan. Jika variabel dan *method* dari kelas pemilik ingin diwariskan untuk kelas turunan (*sub class*) dari kelas pemilik (anak dari kelas pemilik), maka bisa menggunakan hak akses ini. Pewarisan variabel maupun *method* hanya berlaku untuk kelas turunannya dan tidak dapat digunakan pada kelas lain. Pada kelas diagram, hak akses ini diwakili dengan simbol *hashtag* (#).

2. Pewarisan (*Inheritance*)

Sifat ini memungkinkan sebuah kelas menurunkan variabel maupun *method* kepada kelas turunannya. Konsep ini memungkinkan

penggunaan kembali perintah-perintah yang ada di dalam sebuah kelas. Dalam konsep ini dikenal 2 istilah yaitu *parent class* yang biasa disebut *super class* dan *child class* yang biasa disebut *sub class*. Ilustrasi dari pewarisan dapat dilihat pada gambar 8.2 berikut.



Gambar 8.2 Ilustrasi Pewarisan

Pada gambar 8.2 atribut “nama”, “tglLahir”, dan “jnsKelamin” akan diwariskan dari kelas Orang ke kelas “Mahasiswa” dan kelas “Dosen”.

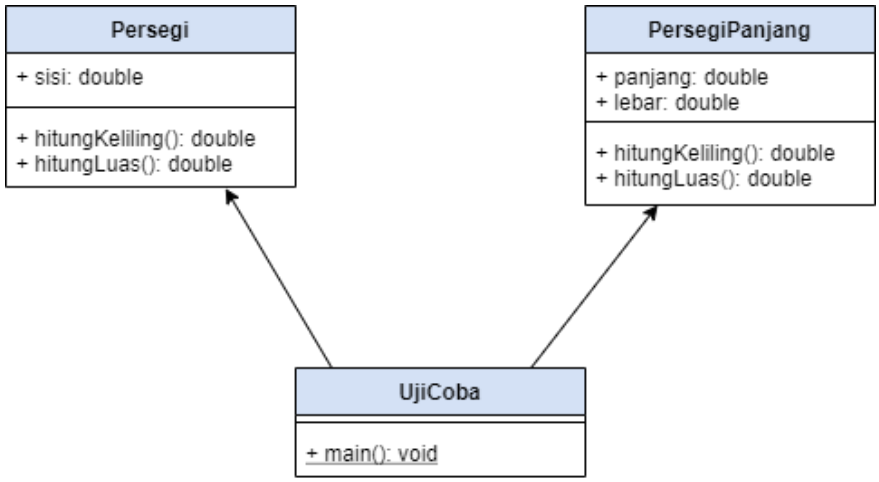
3. Polimorfisme (*Polimorphisme*)

Sifat ini memungkinkan *method* yang diwariskan dari *super class* mempunyai isi perintah yang berbeda ketika diimplementasi pada *sub class*. Sebagai contoh, kelas “Hewan” mempunyai *method* “bersuara()”, maka ketika *method* tersebut diwariskan ke kelas “Kucing” akan berisi meong, dan ketika diwariskan ke kelas “Bebek” akan berisi kwek-kwek.

8.2 Penerapan PBO

1. Penerapan Enkapsulasi dan Instansiasi Objek

Enkapsulasi memastikan atribut dan *method* dibungkus dalam sebuah kelas, dan pemberian hak akses kepada atribut dan *method* hal yang perlu dilakukan. Untuk pemahaman lebih lanjut, maka dibuatlah contoh proyek menggunakan NetBeans IDE yang akan membuat 3 kelas yaitu kelas **Persegi**, **PersegiPanjang**, dan **UjiCoba**. Desain diagram kelas dapat dilihat pada gambar 8.3 berikut.



Gambar 8.3 Diagram kelas contoh enkapsulasi

Pada contoh gambar 8.3 dapat dilihat bahwa *method* "main()" pada kelas "UjiCoba" digaris bawahi, maksudnya adalah bahwa atribut atau *method* tersebut bersifat ***static***.

Kemudian buatlah proyek baru di NetBeans dengan nama "**Contoh_Enkapsulasi**", dan berikutnya buatlah masing-masing kelas sesuai dengan digram kelas sebelumnya.

a. Kelas Persegi

Isikan kode program pada file **Persegi.java** dengan kode program seperti berikut.

```
1  ...5 lines
6  package contoh_enkapsulasi;
7
8  /**...4 lines */
12 public class Persegi {
13     public double sisi;
14
15     public double hitungkeliling() {
16         return 4*sisi; //rumus keliling persegi
17     }
18     public double hitungLuas() {
19         return sisi*sisi; //rumus luas persegi
20     }
21 }
22
```

b. Kelas PersegiPanjang

Isikan kode program pada file **PersegiPanjang.java** dengan kode program seperti berikut.

```
1  ...5 lines
6
7  /**...4 lines */
11 public class PersegiPanjang {
12     public double panjang;
13     public double lebar;
14
15     public double hitungKeliling() {
16         return (2*panjang) + (2*lebar); //rumus keliling persegiPanjang
17     }
18     public double hitungLuas() {
19         return panjang*lebar; //rumus luas persegiPanjang
20     }
21 }
22
```

Setelah membuat 2 kelas yaitu **Persegi** dan **PersegiPanjang** dan mengisikan kode programnya, 2 kelas tersebut belum bisa untuk langsung dijalankan. Mengingat konsep objek dan kelas, sebuah objek hanyalah *template* dari sebuah kelas dan untuk bisa menggunakannya maka perlu dibuat instansiasi yaitu objek. Untuk membuat objek dari sebuah kelas perintahnya seperti berikut:

```
<nama_kelas> <nama_objek> = new
<nama_kelas>();
```

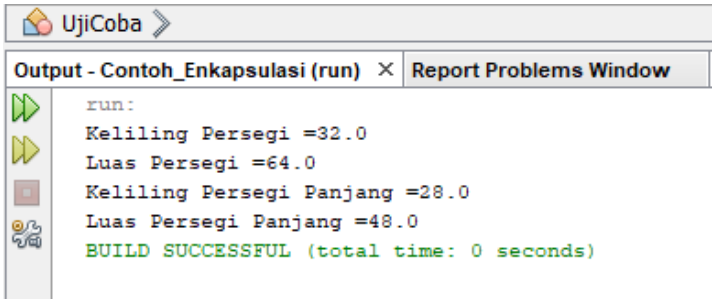
Berdasarkan format tersebut, maka untuk bisa menjalankan *method* yang ada pada kelas **Persegi** dan **PersegiPanjang** perlu dibuat masing-masing objek dari kelas tersebut.

c. Kelas UjiCoba

Isikan kode program pada file **UjiCoba.java** dengan kode program seperti berikut.

```
6
7   /**...4 lines */
11  public class UjiCoba {
12
13      public static void main(String[] args) {
14          Persegi persegi = new Persegi();
15          persegi.sisi = 8;
16          System.out.println("Keliling Persegi ="
17                  + persegi.hitungKeliling());
18          System.out.println("Luas Persegi ="
19                  + persegi.hitungLuas());
20
21          PersegiPanjang persegipanjang = new PersegiPanjang();
22          persegipanjang.panjang = 6;
23          persegipanjang.lebar = 8;
24          System.out.println("Keliling Persegi Panjang ="
25                  + persegipanjang.hitungKeliling());
26          System.out.println("Luas Persegi Panjang ="
27                  + persegipanjang.hitungLuas());
28      }
29  }
```

d. Hasil luaran ketika program dijalankan.



2. Penerapan Getter dan Setter

Setelah berhasil membuat contoh kelas dan objek maka ada 1 prinsip dalam pemrograman berorientasi objek yaitu ketika membuat atribut jangan memberikan hak akses *public* kepada atribut tersebut. Selain

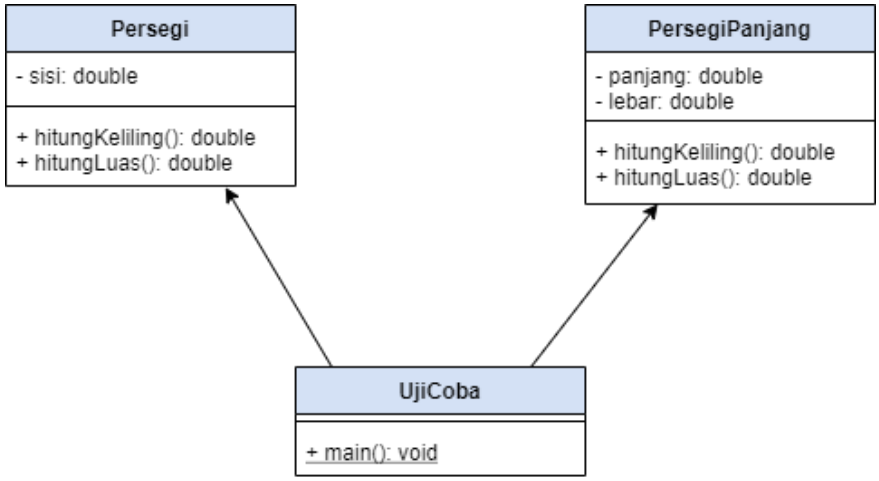
dalam kasus pewarisan (*inheritance*), maka atribut diberikan hak akses *private*. Tetapi, hak akses *private* tidak bisa diakses oleh selain kelas pemilik atribut tersebut.

Pada contoh kelas diagram sebelumnya pada Gambar 8.3, jika mengubah hak akses dari atribut di 2 kelas tersebut menjadi *private* maka tidak bisa mengakses atributnya. Untuk membuat atribut yang mempunyai hak akses *private* menjadi *accesible* (dapat diakses) di kelas selain kelas pemilik, terdapat konsep yang dapat memungkinkan hal tersebut yaitu **Getter** dan **Setter**.

Getter dan Setter merupakan 2 metode yang dibuat untuk masing-masing atribut supaya kelas lain bisa melakukan pengaksesan dan modifikasi nilai yang ada pada suatu atribut. Getter dan Setter mempunyai hak akses *public* sehingga dapat diakses oleh kelas lain.

Getter merupakan *method* yang dibuat untuk mengambil atau mengembalikan nilai sebuah atribut kepada pemanggilnya, sedangkan **Setter** merupakan *method* yang dibuat untuk memodifikasi nilai dari suatu atribut berdasarkan nilai yang dikirim oleh pemanggilnya menggunakan parameter *input*.

Ilustrasi dari penjelasan tersbut adalah seperti gambar 8.4 berikut.



Gambar 8.4 Contoh hak akses dari atribut *private*

Format Getter dan Setter sebagai berikut:

```
//format Getter
public <tipe_data_atribut> getNamaAtribut() {
    return <nama_atribut>;
}
//format Setter
public void
setNamaAtribut(<parameter_input_atribut>) {
    this.<nama_atribut> = <parameter_input_atribut>;
}
```

Kemudian buatlah proyek baru di NetBeans dengan nama "**Getter_Setter**", dan berikutnya buatlah masing-masing kelas sesuai dengan digram kelas sebelumnya.

a. Kelas Persegi

Isikan kode program pada file **Persegi.java** dengan kode program seperti berikut.

```
Source  History
1   ...5 lines
6
7   /**...4 lines */
11  public class Persegi {
12      private double sisi;
13
14      public double getSisi() {
15          return sisi;
16      }
17
18      public void setSisi(double sisi) {
19          this.sisi = sisi;
20      }
21
22      public double hitungKeliling() {
23          return 4*sisi;
24      }
25
26      public double hitungLuas() {
27          return sisi*sisi;
28      }
29  }
```

b. Kelas Persegi Panjang

Isikan kode program pada file **PersegiPanjang.java** dengan kode program seperti berikut.

```
Source  History
 6
 7  /**...4 lines */
11  public class PersegiPanjang {
12      private double panjang;
13      private double lebar;
14
15      public double getPanjang() {
16          return panjang;
17      }
18      public void setPanjang(double panjang) {
19          this.panjang = panjang;
20      }
21
22      public double getLebar() {
23          return lebar;
24      }
25      public void setLebar(double lebar) {
26          this.lebar = lebar;
27      }
28
29      public double hitungKeliling() {
30          return (2*panjang) + (2*lebar);
31      }
32      public double hitungLuas() {
33          return panjang*lebar;
34      }
35  }
```

Dapat dilihat pada kelas "**Persegi**" dan "**PersegiPanjang**" terdapat Getter dan Setter yang ditambahkan. Karena berbentuk *method*, pemanggilannya dipanggil dengan cara yang sama seperti pemanggilan *method* pada umumnya.

c. Kelas UjiCoba

Isikan kode program pada file **UjiCoba.java** dengan kode program seperti berikut.

```
Source  History
1  ...5 lines
6
7  /**...4 lines */
11 public class UjiCoba {
12     public static void main(String[] args) {
13         Persegi persegi = new Persegi();
14         persegi.setSisi(8);
15         System.out.println("Sisi Persegi ="
16                 + persegi.getSisi());
17         System.out.println("Keliling Persegi ="
18                 + persegi.hitungKeliling());
19         System.out.println("Luas Persegi ="
20                 + persegi.hitungLuas());
21
22         PersegiPanjang persegipanjang = new PersegiPanjang();
23         persegipanjang.setPanjang(8);
24         persegipanjang.setLebar(6);
25         System.out.println("Panjang PersegiPanjang ="
26                 + persegipanjang.getPanjang());
27         System.out.println("Lebar PersegiPanjang ="
28                 + persegipanjang.getLebar());
29         System.out.println("Keliling PersegiPanjang ="
30                 + persegipanjang.hitungKeliling());
31         System.out.println("Luas PersegiPanjang ="
32                 + persegipanjang.hitungLuas());
33     }
34 }
```

d. Hasil luaran ketika program dijalankan.

```
Output - Getter_Setter (run)  ×  Report Problems Window
run:
Sisi Persegi =8.0
Keliling Persegi =32.0
Luas Persegi =64.0
Panjang PersegiPanjang =8.0
Lebar PersegiPanjang =6.0
Keliling PersegiPanjang =28.0
Luas PersegiPanjang =48.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Penerapan Konstruktor dan Destruktor

Konstruktor dibuat untuk menandakan bahwa objek dari suatu kelas sedang diinstansiasi, sementara destruktor menandakan bahwa objek dihancurkan dari memori. Konstruktor dan destruktor merupakan *method* khusus yang tidak mempunyai tipe data. Dalam Java terdapat konsep "***Garbage Collector***" sehingga JDK akan secara otomatis akan menghancurkan objek yang dianggap sudah tidak digunakan. Oleh karena itu di Java cukup membuat konstruktor secara eksplisit, sementara destruktor bersifat opsional.

Konstruktor di Java harus diberikan nama yang sama dengan nama kelasnya. Konstruktor akan dipanggil secara otomatis ketika menginstansiasi sebuah atau lebih objek dari kelas yang memiliki konstruktor tersebut. Biasanya selain untuk menandakan instansiasi objek dari suatu kelas, konstruktor digunakan untuk memberikan nilai standar (nilai *default*) atau nilai awal pada atribut yang terdapat pada suatu kelas.

Untuk memberikan gambaran yang lebih jelas maka akan digunakan proyek sebelumnya untuk digunakan sebagai contoh.

a. Buka proyek sebelumnya "**Getter_Setter**".

b. Lakukan penambahan perintah konstruktor pada bagian bawah atribut di kelas “**Persegi**” dan “**PersegiPanjang**”.

Persegi.java

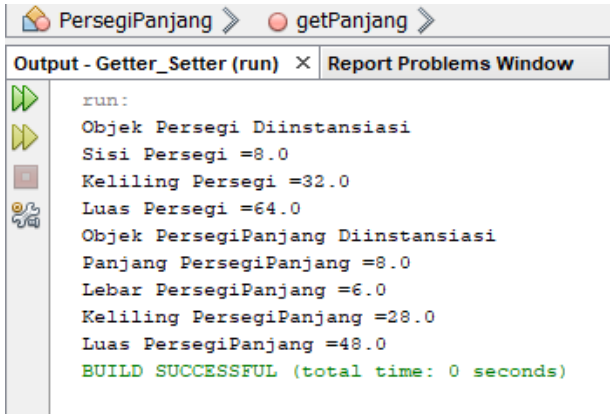
```
/**...4 lines */
public class Persegi {
    private double sisi;
    //bagian konstruktor
    public Persegi(){
        this.sisi = 0;
        System.out.println("Objek Persegi Diinstansiasi");
    }

    public double getSisi() {
        return sisi;
    }
    public void setSisi(double sisi) {
        this.sisi = sisi;
    }

    public double hitungKeliling() {
        return 4*sisi;
    }
    public double hitungLuas() {
        return sisi*sisi;
    }
}
```

PersegiPanjang.java

```
...5 lines

/**...4 lines */
public class PersegiPanjang {
    private double panjang;
    private double lebar;

    //bagian konstruktor
    public PersegiPanjang() {
        this.panjang = 0;
        this.lebar = 0;
        System.out.println("Objek PersegiPanjang Diinstansiasi");
    }

    public double getPanjang() {
        return panjang;
    }
```

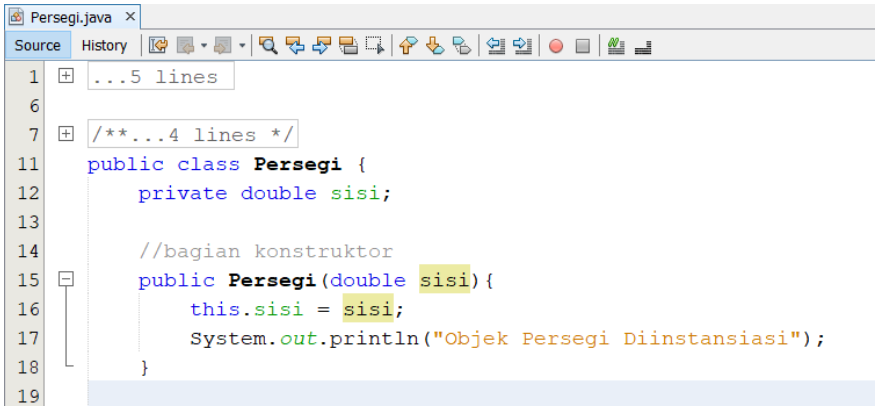
c. Pada kelas "**UjiCoba**" tidak dilakukan perubahan dikarenakan pemanggilan konstruktor tidak membutuhkan perintah baru. Konstruktor akan dipanggil secara otomatis ketika instansiasi objek dilakukan.

d. Hasil luaran ketika program dijalankan.

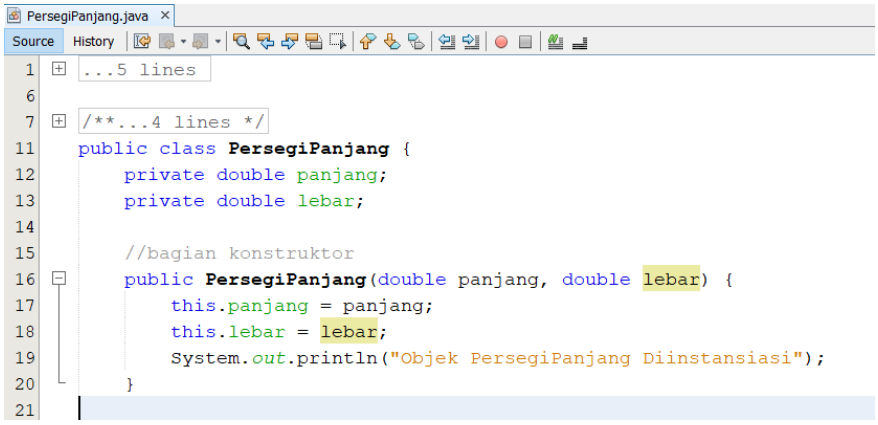


Selain memberikan nilai standar pada atribut, konstruktor juga dapat digunakan untuk memberikan nilai awal atau biasa disebut sebagai menginisialisasi nilai pada suatu atribut di dalam kelas. Contoh gambaran seperti berikut:

a. Kelas Persegi



b. Kelas PersegiPanjang



c. Kelas UjiCoba

Karena konstruktor yang dibuat mengandung parameter *input*, maka proses instansiasi objek kelas "**UjiCoba**" juga harus menyertakan niulai untuk parameter.

UjiCoba.java

Source | History

```
7   /**...4 lines */
11  public class UjiCoba {
12      public static void main(String[] args) {
13          Persegi persegi = new Persegi(8);
14          System.out.println("Sisi Persegi ="
15                  + persegi.getSisi());
16          System.out.println("Keliling Persegi ="
17                  + persegi.hitungKeliling());
18          System.out.println("Luas Persegi ="
19                  + persegi.hitungLuas());
20
21          PersegiPanjang persegipanjang = new PersegiPanjang(8, 6);
22          System.out.println("Panjang PersegiPanjang ="
23                  + persegipanjang.getPanjang());
24          System.out.println("Lebar PersegiPanjang ="
25                  + persegipanjang.getLebar());
26          System.out.println("Keliling PersegiPanjang ="
27                  + persegipanjang.hitungKeliling());
28          System.out.println("Luas PersegiPanjang ="
29                  + persegipanjang.hitungLuas());
30      }
31  }
```

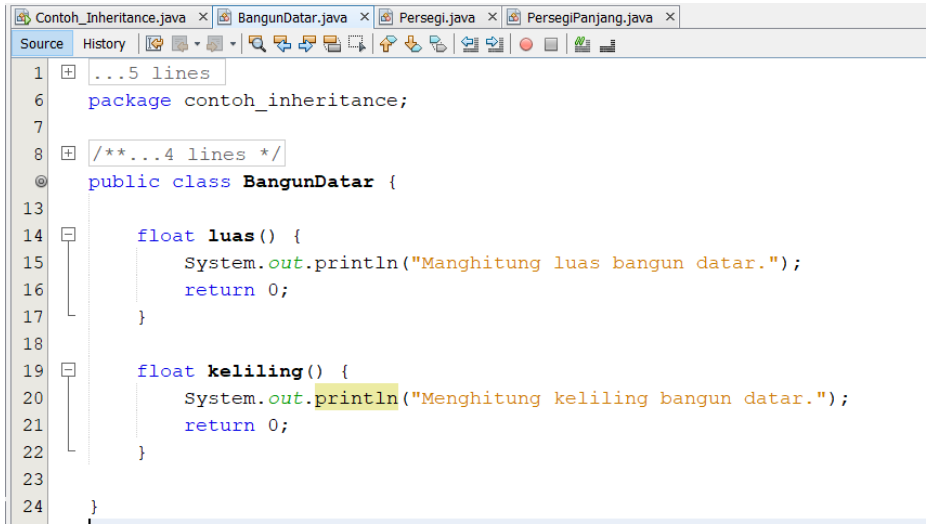
d. Luaran dari kelas "**UjiCoba**" ketika dijalankan akan sama dengan hasil sebelumnya, tetapi yang membedakan adalah pada baris 13 dan 21 yang berisi perintah instansiasi objek mempunyai nilai yang akan menjadi parameter *input* di dalam konstruktor yang dibuat.

4. Penerapan Pewarisan (*Inheritance*)

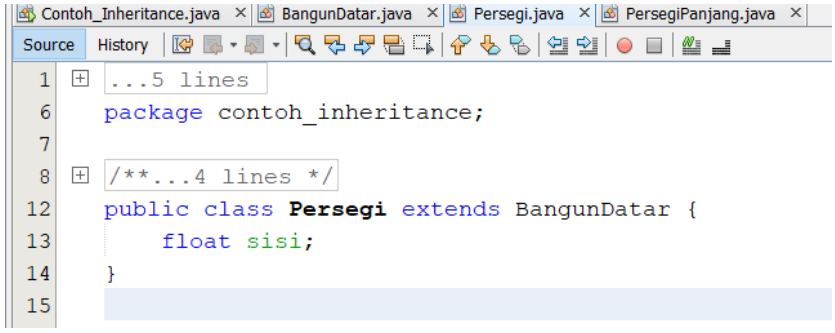
Buatlah projek baru di NetBeans dengan nama "**Contoh_Inheritance**", program yang akan dibuat ialah untuk menghitung luas dan keliling bangun datar (persegi dan persegi panjang). Dalam projek ini buatlah tiga kelas (*class*) baru yaitu **BangunDatar**, **Persegi**, dan **PersegiPanjang**.

Berikut isian kode program pada masing-masing kelas:

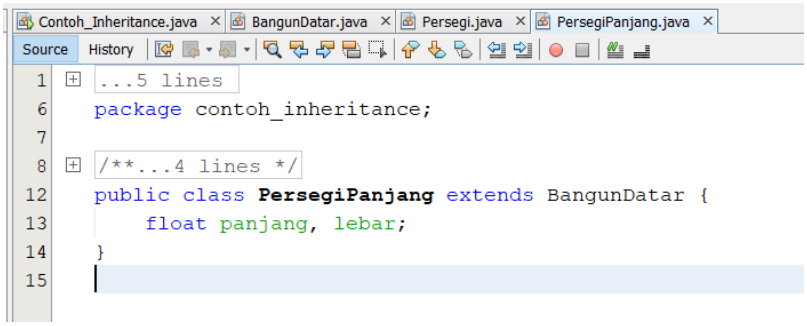
BangunDatar.java



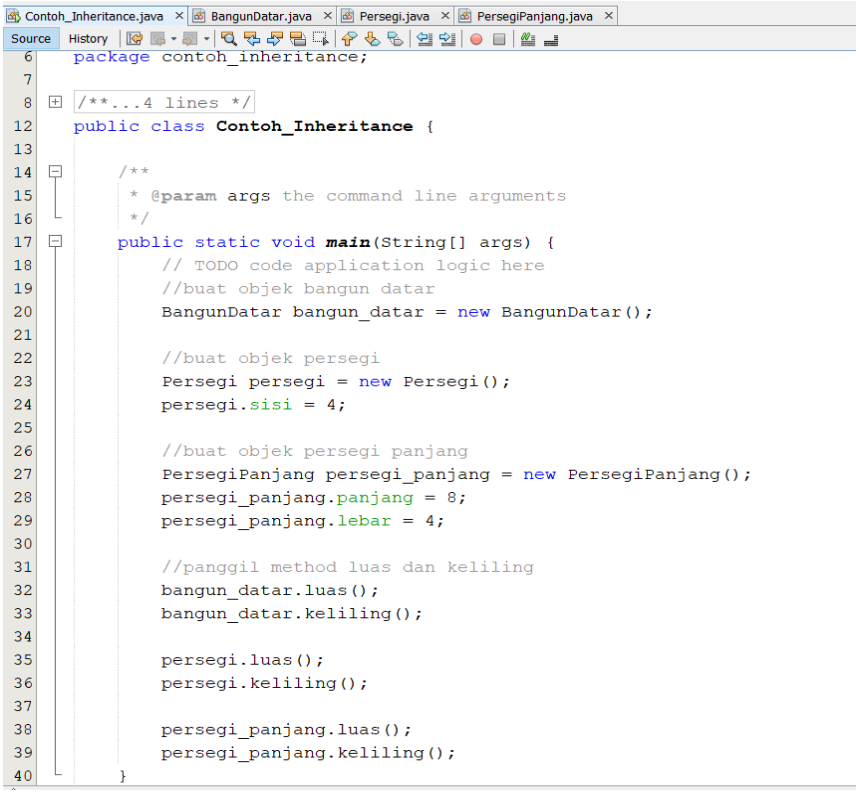
Persegi.java



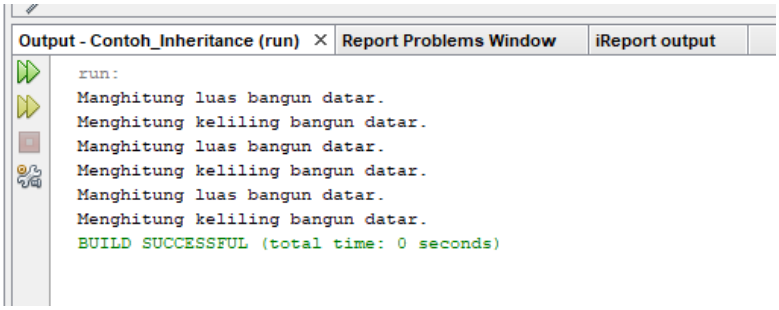
PersegiPanjang.java



Contoh_Inheritance.java



Kemudian jika program dijalankan, hasilnya seperti berikut:



Kenapa bisa seperti itu hasilnya?

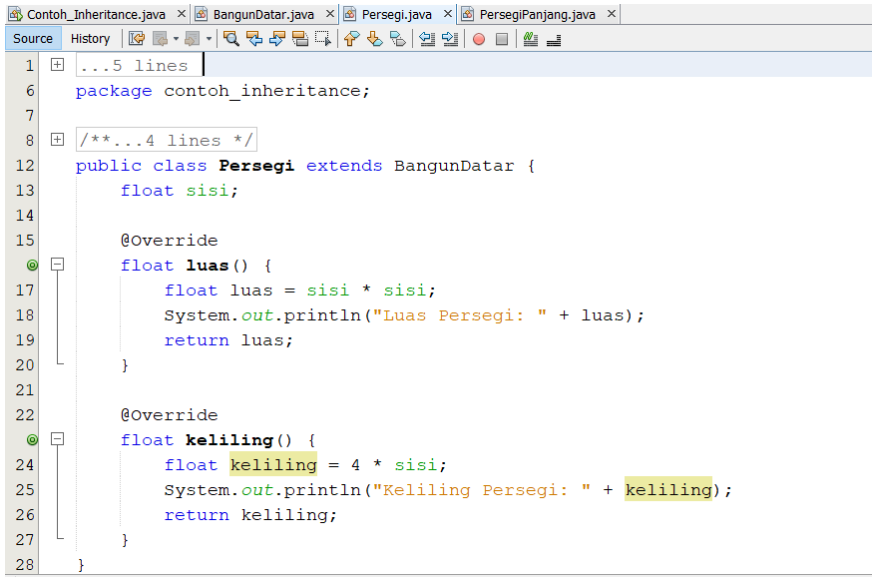
Karena yang dipanggil sebenarnya ialah method **luas()** dan **keliling()** milik induk **BangunDatar**. Objek anak dari **BangunDatar** belum memiliki method **luas()** dan **keliling()**, maka mengambil method dari induknya. Kemudian bagaimana kalau ingin membuat agar semua kelas anak mempunyai method **luas()** dan **keliling()** yang berbeda dari induk?

Yaitu menggunakan method *overriding*.

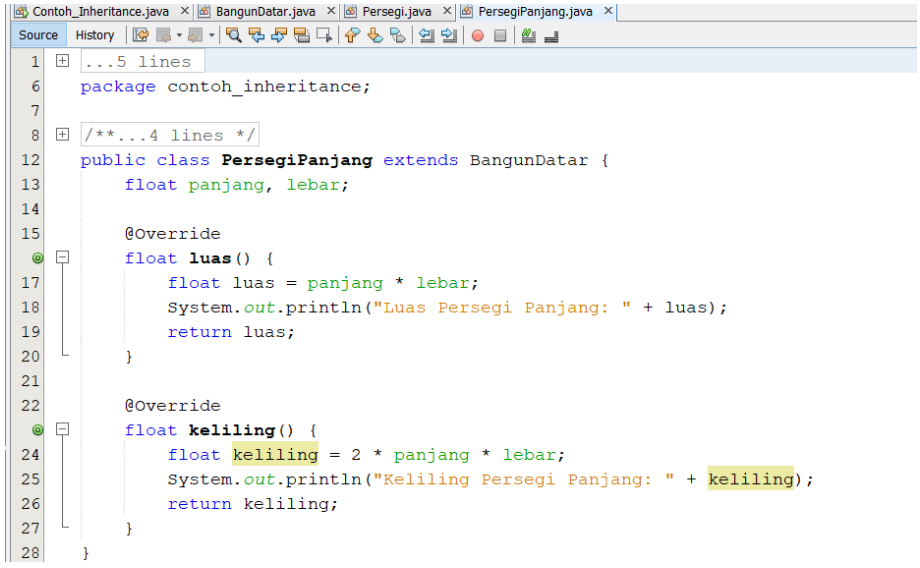
Method Overriding

Metode ini dilakukan ketika ingin membuat ulang sebuah method pada *sub-class* atau *class* anak. Method *overriding* dapat dibuat dengan menambahkan anotasi **@Override** sebelum pembuatan method. Tambahkan kode program pada kelas **Persegi** dan **PersegiPanjang** seperti berikut:

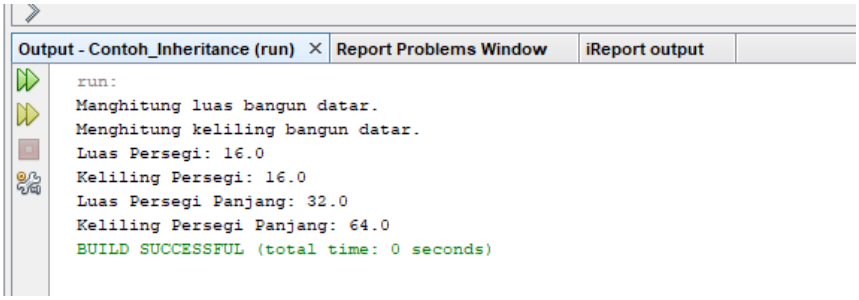
Kelas **Persegi**.



Kelas **PersegiPanjang**.



Kemudian coba jalankan programnya.



5. Polimorfisme

Polimorfisme dalam OOP merupakan sebuah prinsip dimana *class* dapat mempunyai banyak bentuk method yang berbeda-beda meskipun namanya sama. Bentuk dalam hal ini dapat dipahami seperti isi yang berbeda, parameter yang berbeda, dan tipe data yang berbeda. Dalam Java terdapat dua macam polimorfisme yaitu **Polimorfisme Statis** dan **Polimorfisme Dinamis**. Perbedaan dari keduanya ialah terletak pada cara pembuatannya, polimorfisme statis menggunakan ***method overloading*** sedangkan polimorfisme dinamis menggunakan ***method overriding***.

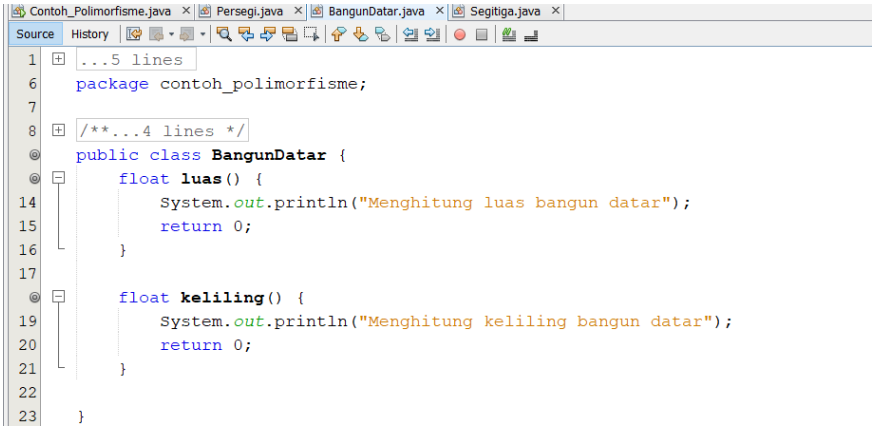
Metode *overloading* terjadi pada sebuah kelas yang mempunyai “nama method” yang sama, tetapi mempunyai “parameter” dan “tipe data” yang berbeda. Jadi polimorfisme statis hanya terjadi dalam satu kelas

saja, sedangkan polimorfisme dinamis terjadi pada saat ada hubungan dengan kelas lainnya seperti *inheritance*.

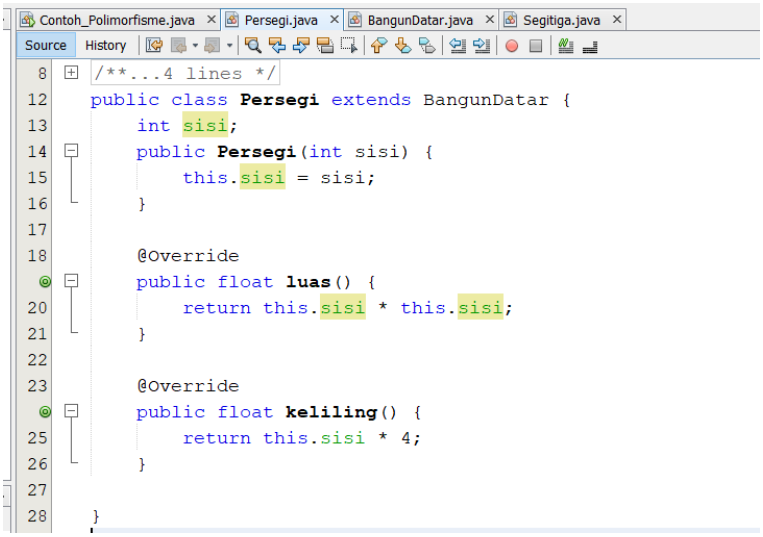
Contoh program hampi sama seperti pada program di *inheritance* yang sudah dibuat sebelumnya, hanya berbeda kasus bangun datarnya. Buatlah projek baru dengan nama **Contoh_Polimorfisme**, kemudian buatlah tiga kelas baru dengan nama **BangunDatar**, **Persegi**, dan **Segitiga**.

Berikut isian kode program pada masing-masing kelas:

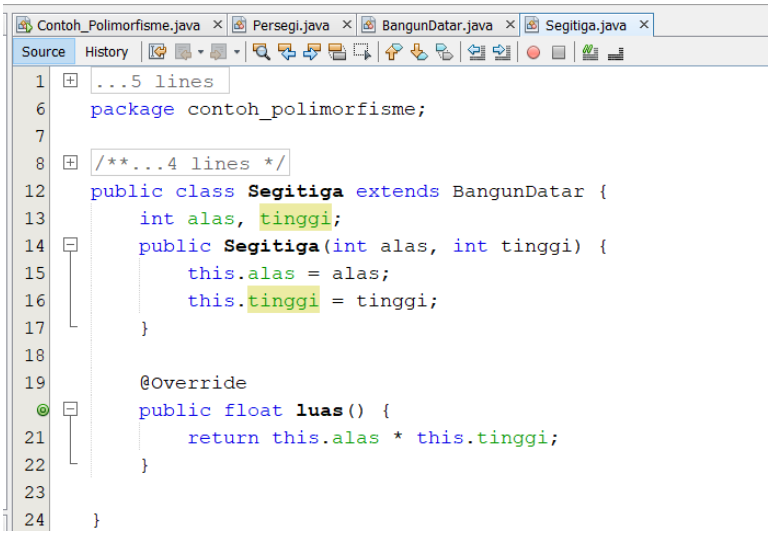
Kelas **BangunDatar**



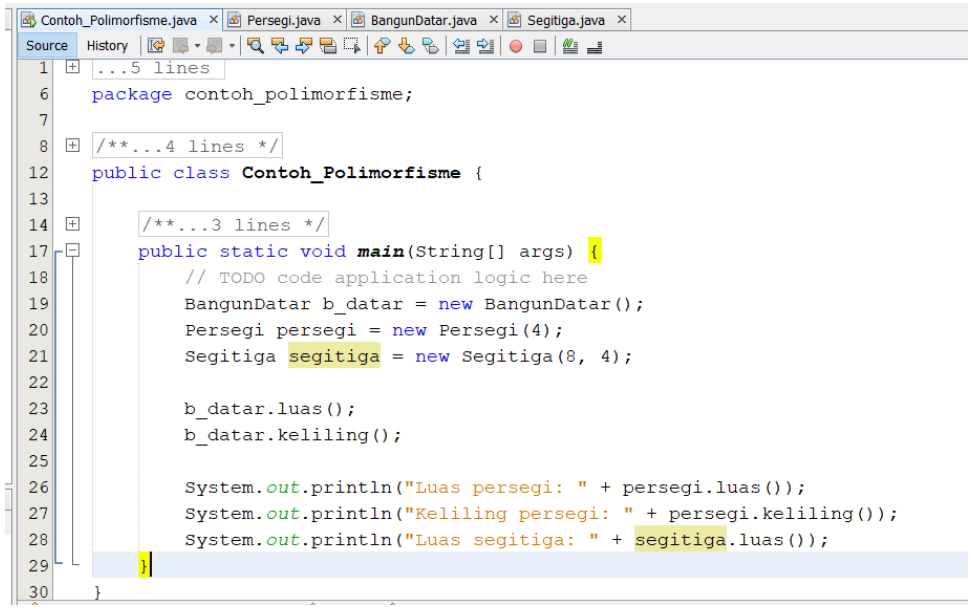
Kelas **Persegi**



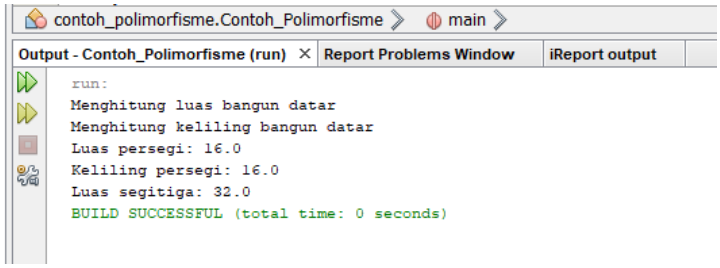
Kelas **Segitiga**



Kelas utama (*main class*) **Contoh_Inheritance**



Berikut hasil program ketika dijalankan:



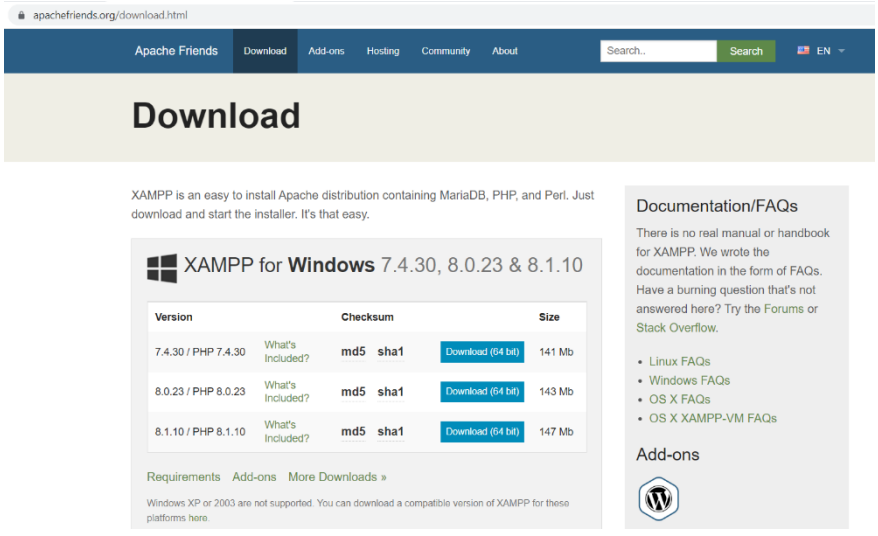
BAB 9
PENGGUNAAN *DATABASE*

9.1 Pembuatan *Database*

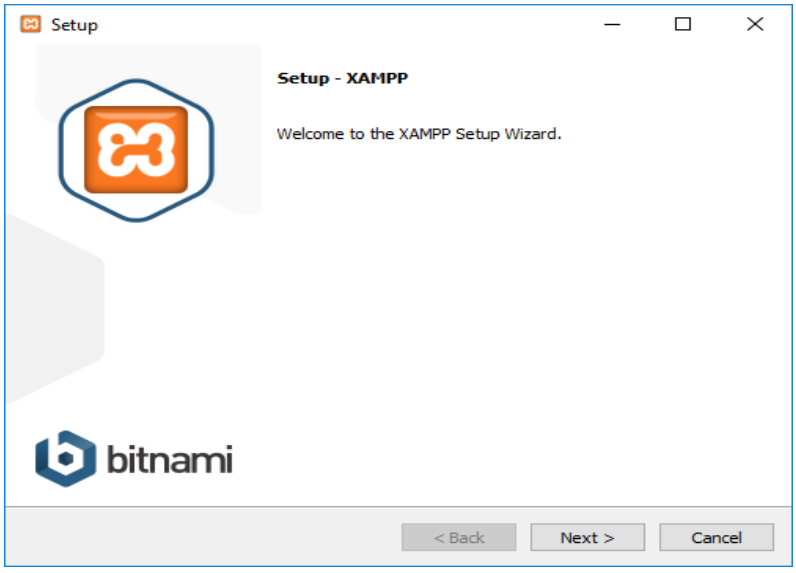
Pada praktik ini akan menggunakan database MySQL yang ada di dalam paket XAMPP. Untuk itu dibutuhkan perangkat lunak tambahan yaitu XAMPP yang akan digunakan sebagai local host dari program sederhana yang akan dibuat.

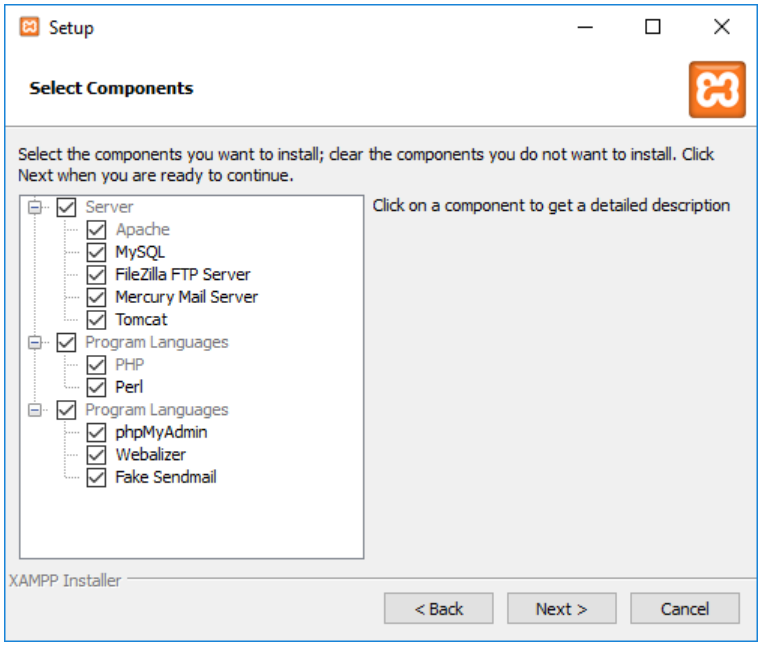
Langkah-langkah dalam pembuatan database yang akan digunakan adalah seperti berikut ini:

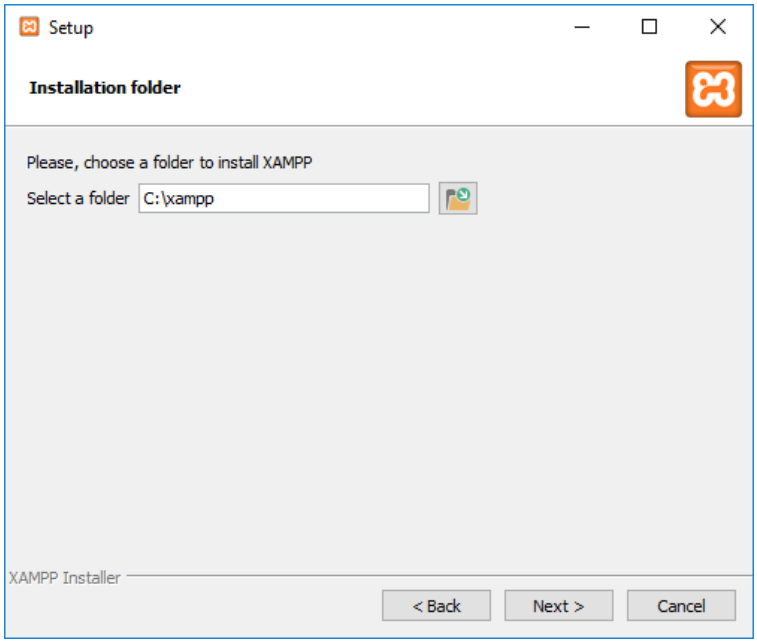
1. Unduh dahulu perangkat lunak XAMPP melalui situs resminya yaitu di https://www.apachefriends.org/download.html secara gratis/*open source*.



2. Setelah file terunduh, lakukan instalasi pada perangkat lunak tersebut.



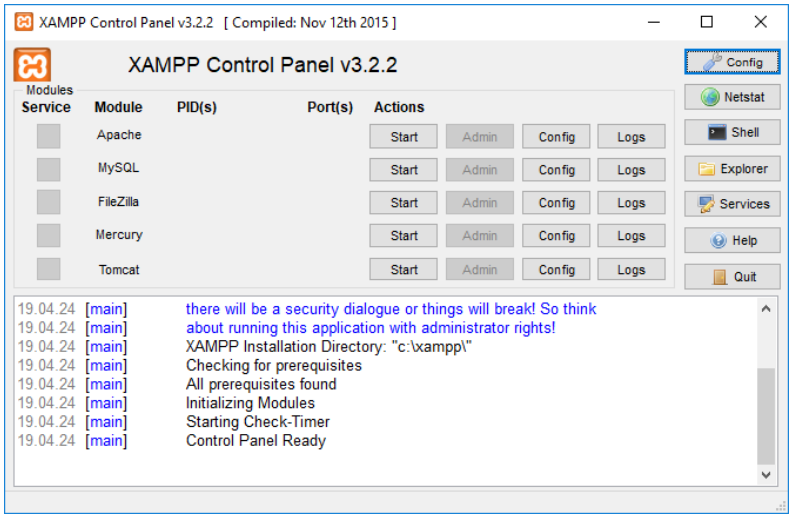




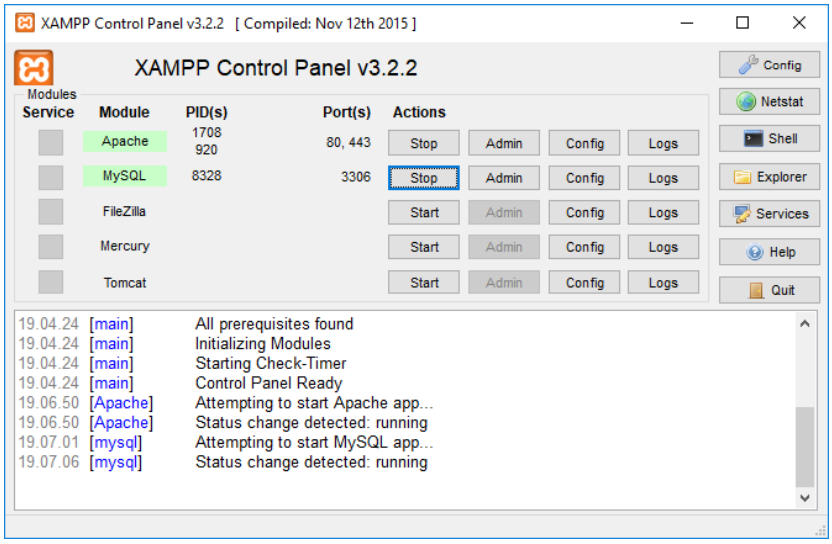
Pilih lokasi/folder instalasi.

Kemudian klik Next hingga proses instalasi selesai.

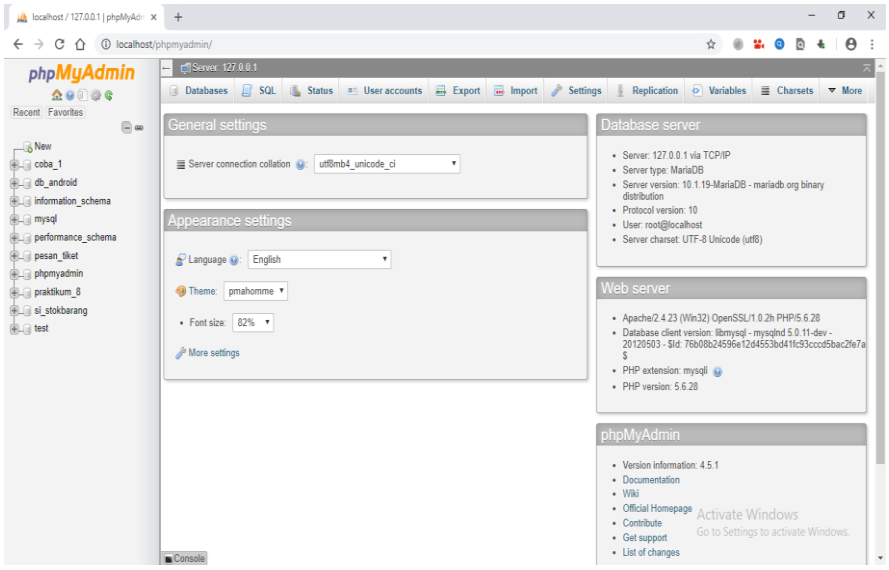
3. Setelah terinstal, buka aplikasi XAMPP.



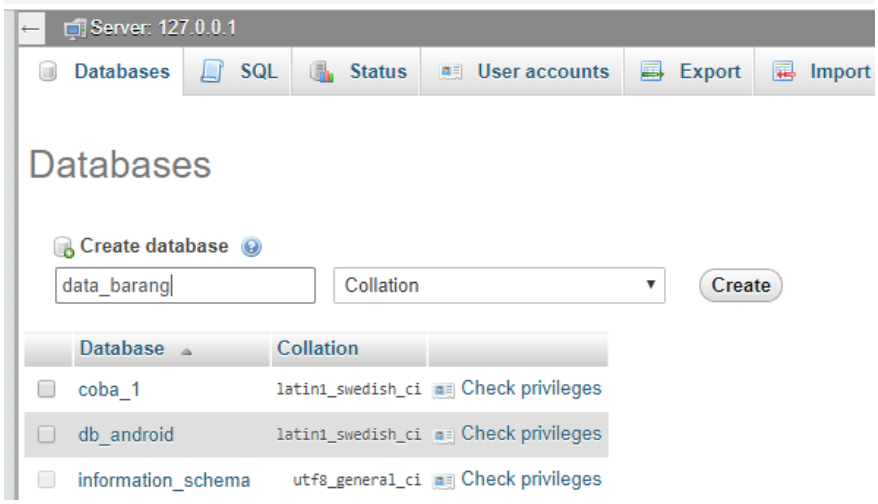
4. Pada tampilan di atas terdapat beberapa modules service, namun dalam praktik ini hanya dibutuhkan dua services yaitu Apache dan MySQL, dimana Apache digunakan sebagai local host dan MySQL digunakan sebagai penampung database. Aktifkan kedua services tersebut dengan cara klik tombol Start.



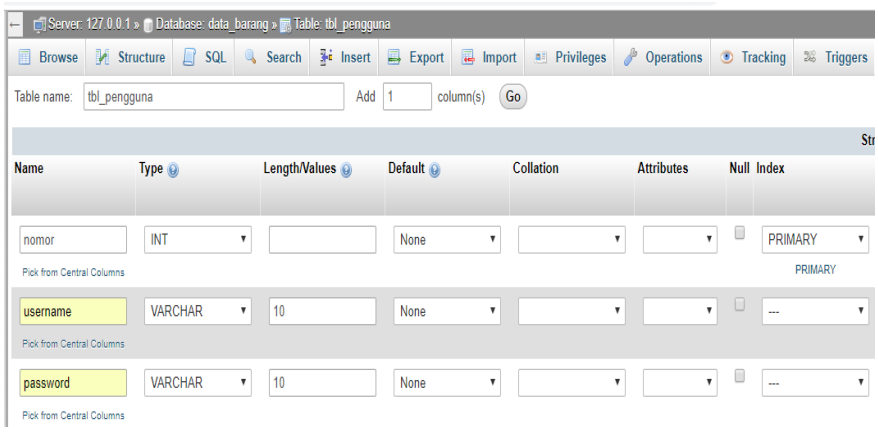
5. Selanjutnya, buka aplikasi browser dan buka alamat **localhost/phpmyadmin/** seperti berikut.



6. Selanjutnya buat database baru dengan cara klik menu Database pada ToolBar isikan nama database "**data_barang**" → klik tombol Create.



7. Selanjutnya buat tabel baru dengan cara Create Table → isikan nama "**tbl_pengguna**" → isikan jumlah kolom (3 kolom) → klik tombol Go.
8. Selanjutnya isikan ketentuan-ketentuan dari masing-masing kolom → klik Save.

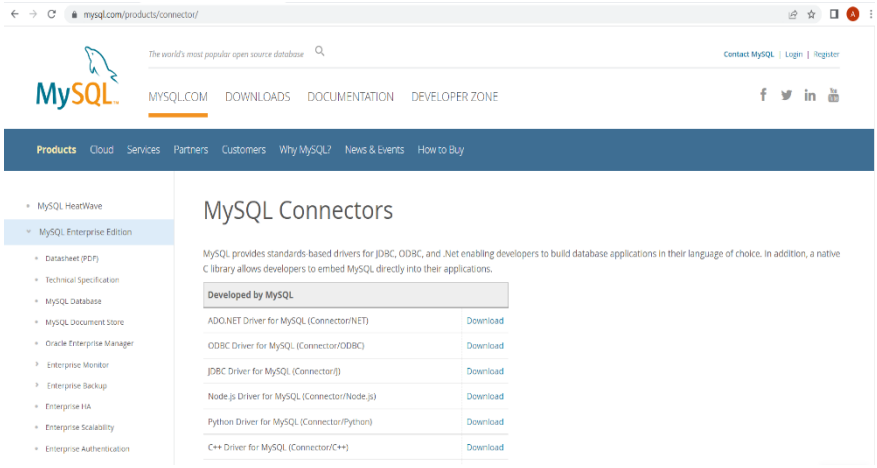


9. Kemudian isikan data pada tabel tersebut.

9.2 Koneksi Database

Pada tahapan ini dilakukan konfigurasi/pengaturan pada NetBeans IDE agar dapat terhubung dengan database MySQL untuk melakukan operasi database yang telah dibuat sebelumnya, yaitu database "**data_barang**". Langkah-langkah pengaturan tersebut seperti berikut ini:

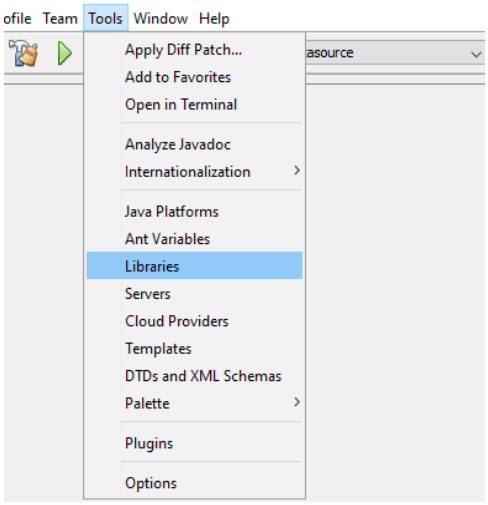
1. Buka NetBeans IDE.
2. Untuk menghubungkan kepada database MySQL, dibutuhkan sebuah pustaka (*library*) untuk mendukung hal tersebut. Pustaka tersebut dapat diunduh secara gratis di situs resminya yaitu https://www.mysql.com/products/connector/ .

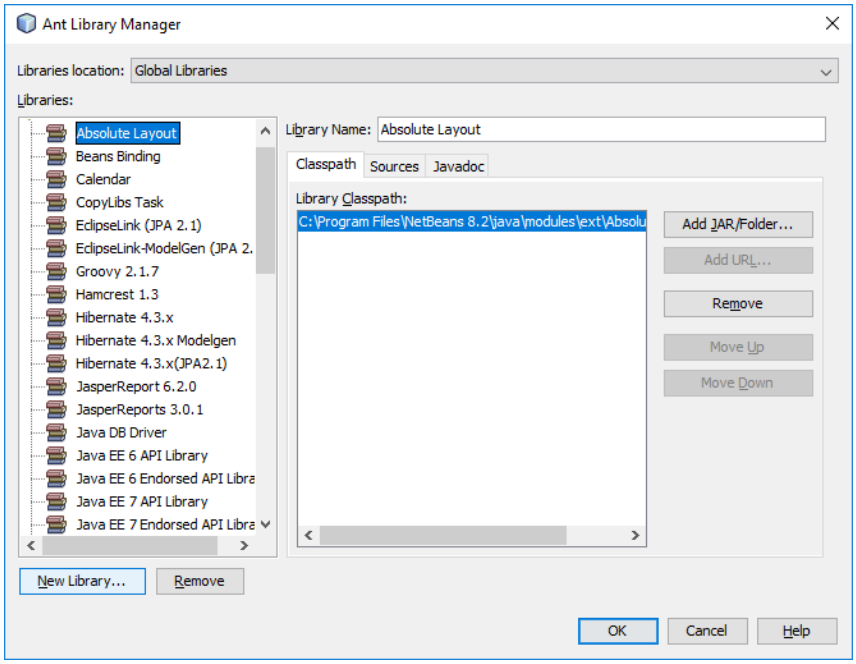


Pada halaman tersebut terdapat beberapa pilihan, pilih file dengan nama **JDBC Driver for MySQL (Connector/J)** → Download.

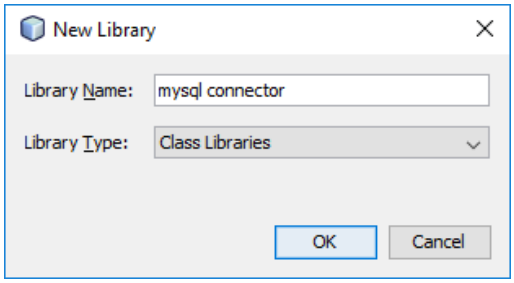
Kemudian ikuti petunjuknya, sesuaikan dengan spesifikasi laptop/PC.

3. Selanjutnya kembali lagi kepada NetBeans IDE, klik menu Tools pada ToolBar → pilih **Libraries**.

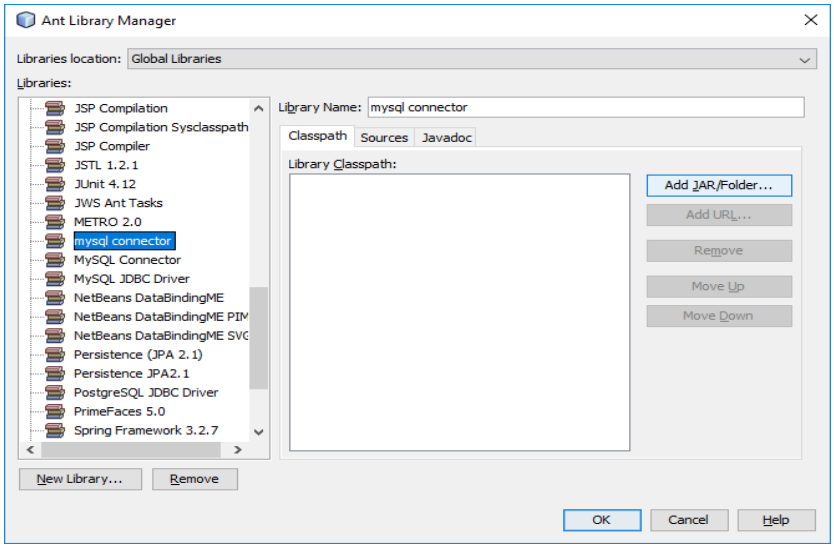




4. Setelah muncul wizard Ant Library Manager → klik tombol New Library → isikan nama library "**mysql connector**" → klik OK.



5. Kemudian klik library yang telah dibuat sebelumnya → klik **Add JAR/Folder** → tambahkan file MySQL Connector yang telah diunduh sebelumnya.

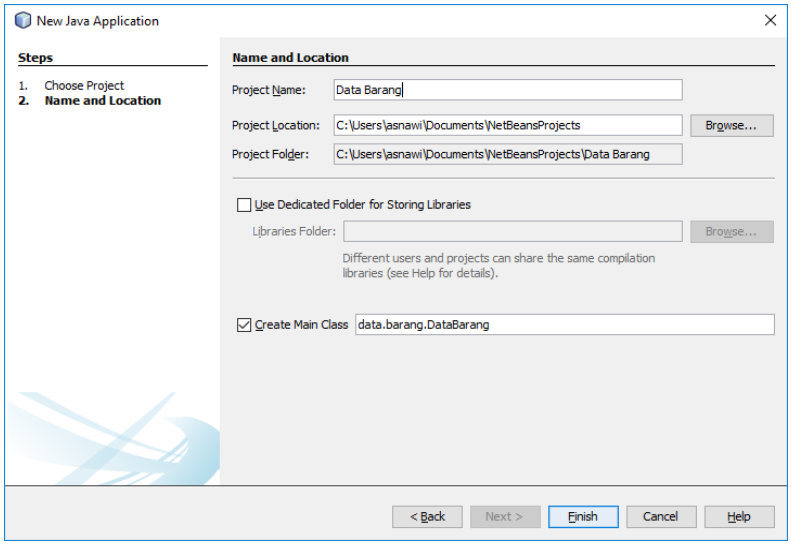


6. Sampai tahapan ini untuk persiapan koneksi ke database telah selesai.

9.1 Operasi Dasar Database

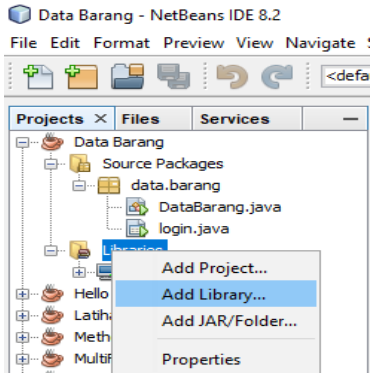
Pada tahapan ini akan membahas mengenai pembuatan program sederhana yang memanfaatkan *database* MySQL. Program tersebut memuat halaman login, halaman menu utama, dan halaman data barang. Dalam program tersebut akan menggunakan operasi dasar database untuk membuat data yang akan disimpan ke dalam database dan membaca data dari database. Program tersebut dibuat dengan langkah-langkah seperti berikut ini:

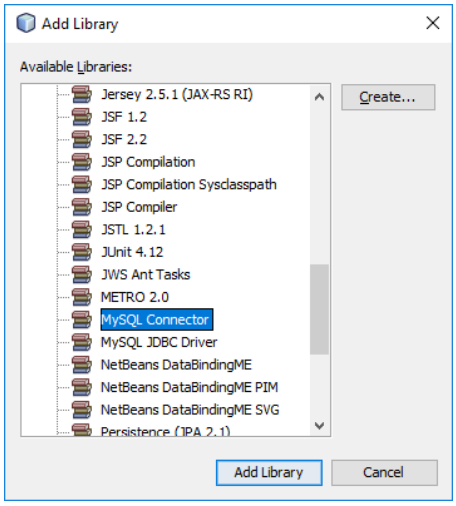
1. Buka NetBeans IDE
2. Buat project baru dengan nama “Data Barang”.

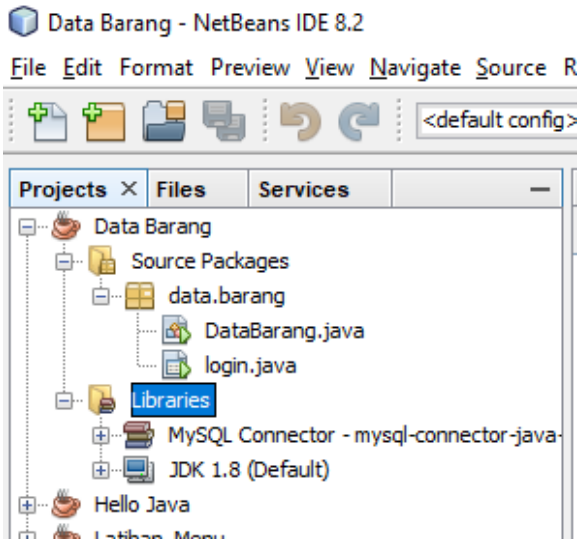


3. Kemudian pada folder Libraries, tambahkan library MySQL connector. Pada library tersebut, terdapat beberapa komponen

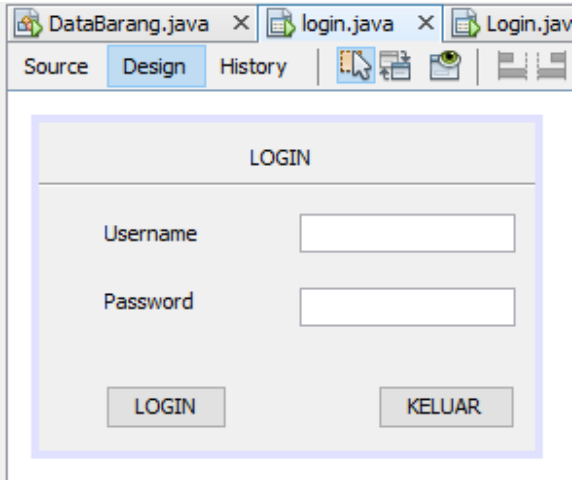
yang akan digunakan dalam kode program yang berhubungan dengan database. Langkah-langkahnya seperti berikut.







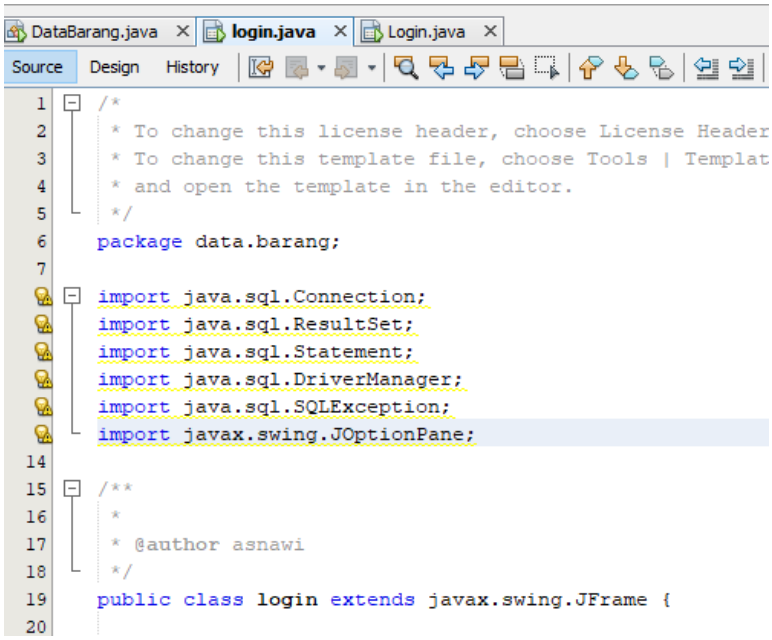
4. Kemudian buat sebuah form dengan tampilan desain seperti berikut.

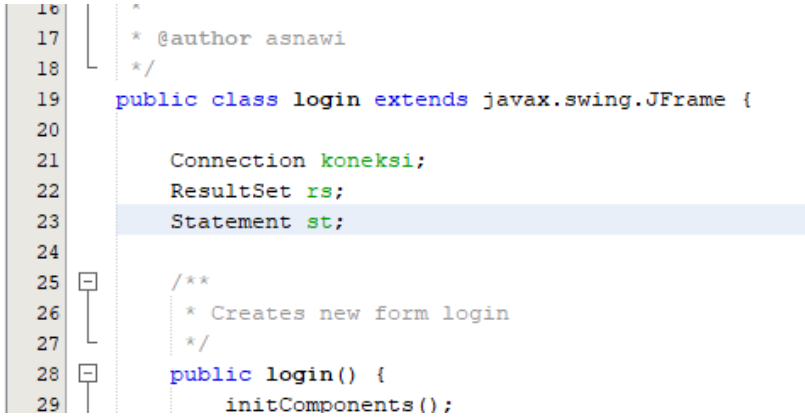


Pada penambahan komponen tersebut terdapat pengaturan yaitu mengubah label button1 menjadi “LOGIN” dan mengubah nama

variabel dengan nama “btnLogin”, pada label button2 menjadi “KELUAR” dan mengubah nama variabel dengan nama “btnKeluar”.

5. Setelah pembuatan form login selesai, kemudian mengisikan kode program pada form login dengan seperti berikut.





DataBarang.java | login.java | Login.java

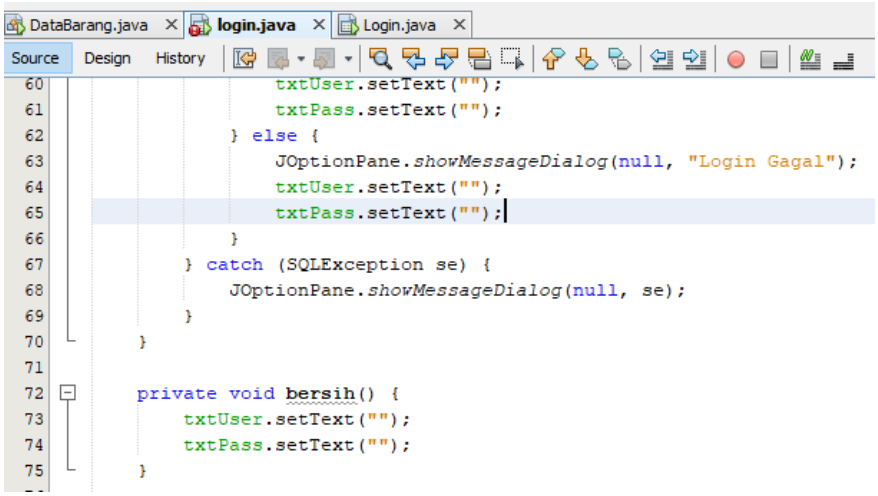
Source | Design | History

```
21      Connection koneksi;
22      ResultSet rs;
23      Statement st;
24
25      /**
26       * Creates new form login
27       */
28      public login() {
29          initComponents();
30          bukaKoneksi();
31      }
32
33      //koneksi ke database
34      private void bukaKoneksi() {
35          try {
36              Class.forName("com.mysql.jdbc.Driver");
37              koneksi = DriverManager.getConnection("jdbc:mysql://localhost/data_barang", "root", "");
38              System.out.println("Koneksi Berhasil");
39          } catch (Exception e) {
40              System.out.println(e);
41          }
42      }
```

DataBarang.java | login.java | Login.java

Source | Design | History

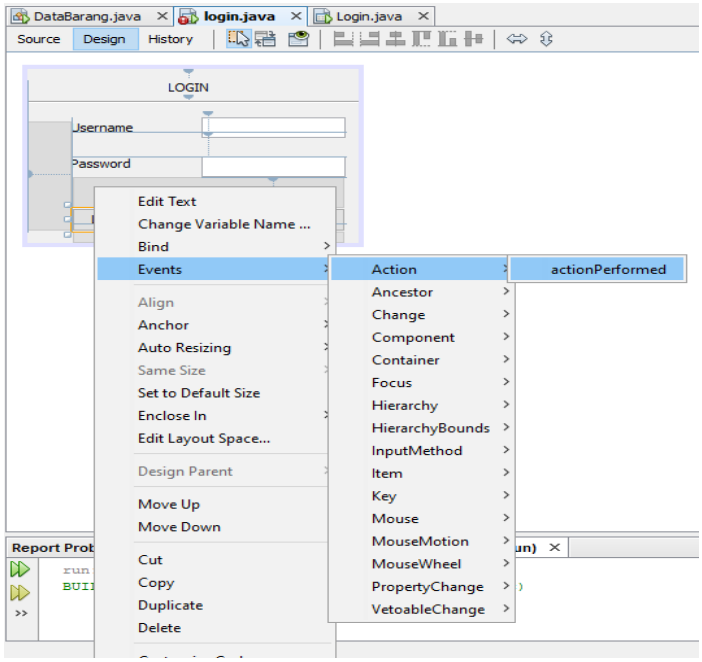
```
44      //login
45      private void login() {
46          String nama = txtUser.getText();
47          String password = new String(txtPass.getPassword());
48          int kode = 0;
49          try {
50              st = koneksi.createStatement();
51              String sql = "SELECT * FROM tbl_pengguna WHERE username='"+nama+"' AND password='"+password+"'";
52              rs = st.executeQuery(sql);
53              int baris = 0;
54              while (rs.next()) {
55                  kode = rs.getInt("nomor");
56                  baris = rs.getRow();
57              }
58              if (baris==1) {
59                  new Menu_Utama().setVisible(true);
60                  txtUser.setText("");
61                  txtPass.setText("");
62              } else {
63                  JOptionPane.showMessageDialog(null, "Login Gagal");
64                  txtUser.setText("");
65                  txtPass.setText("");
66              }
67          } catch (SQLException se) {
```



Dalam mengisikan kode program, **perhatikan tata letaknya**.

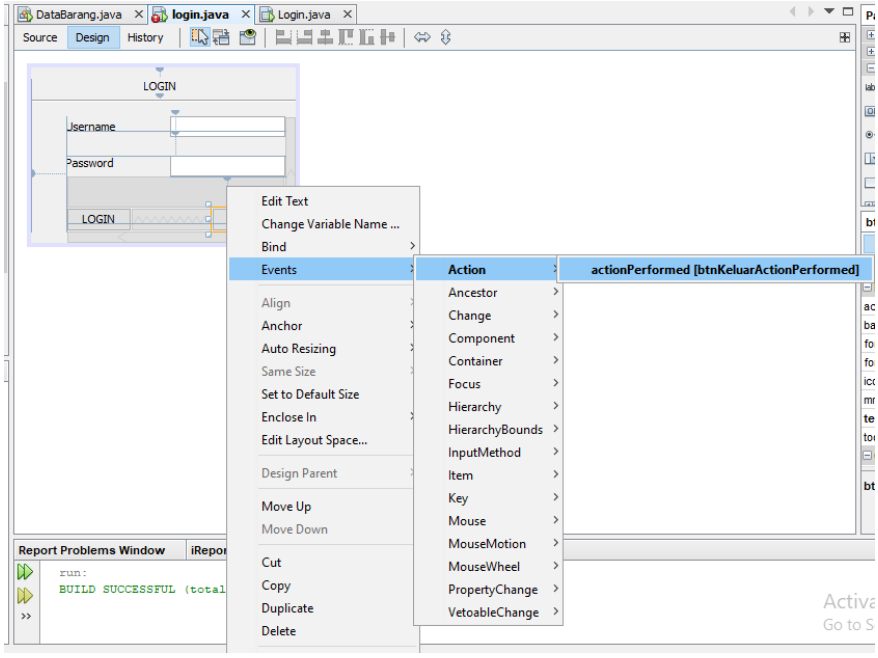
6. Kemudian berikan event kepada tombol LOGIN dan KELUAR seperti berikut.

Tombol LOGIN:



```
81      */
82     @SuppressWarnings("unchecked")
83     Generated Code

168
169     private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
170         // TODO add your handling code here:
171         login();
172     }
173
```

Tombol KELUAR:

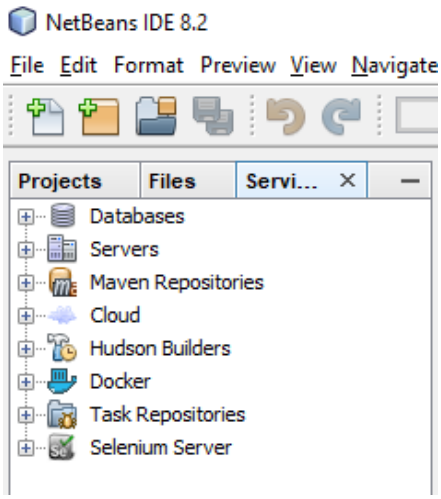



```
174     private void btnKeluarActionPerformed(java.awt.event.ActionEvent evt) {
175         // TODO add your handling code here:
176         System.exit(0);
177     }
178
```

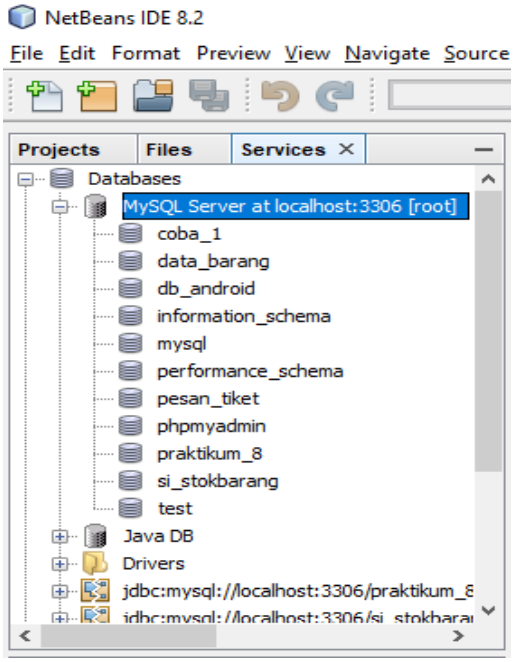
7. Selanjutnya tahap penambahan kode program terakhir yaitu paada class main dari aplikasi, dimana berfungsi untuk menampilkan form yang akan pertama kali dipanggil/dijalankan.

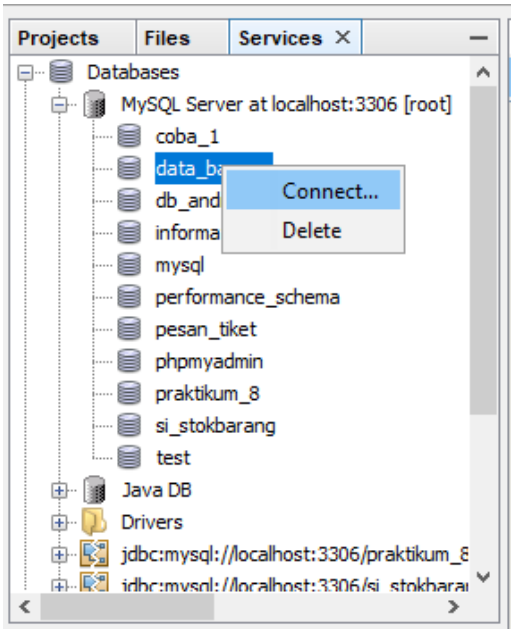
```
10      * @author asnawi
11      */
12     public class DataBarang {
13
14         /**
15          * @param args the command line arguments
16          */
17         public static void main(String[] args) {
18             // TODO code application logic here
19             new login().setVisible(true);
20         }
21
22     }
```

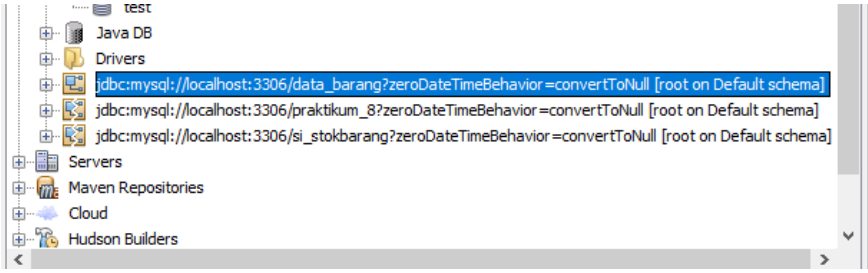
8. Kemudian tambahkan pengaturan pada tab menu **Services**.



9. Buka Database → MySQL Server at localhost → klik kanan databse "**data_barang**" → pilih Connect.

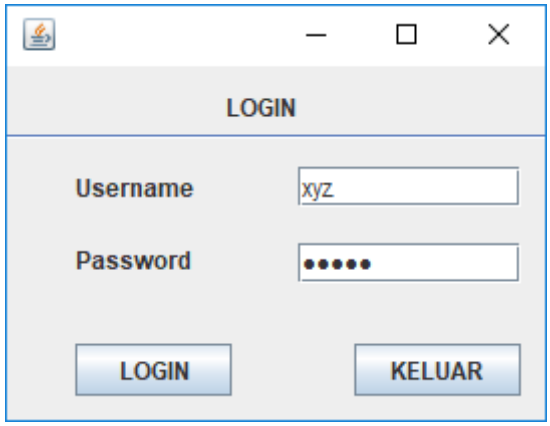

NetBeans IDE 8.2
File Edit Format Preview View Navigate Source
Projects
Files
Services
Databases
MySQL Server at localhost:3306 [root]
coba_1
data_barang
db_android
information_schema
mysql
performance_schema
pesan_tiket
phpmyadmin
praktikum_8
si_stokbarang
test
Java DB
Drivers
jdbc:mysql://localhost:3306/praktikum_8

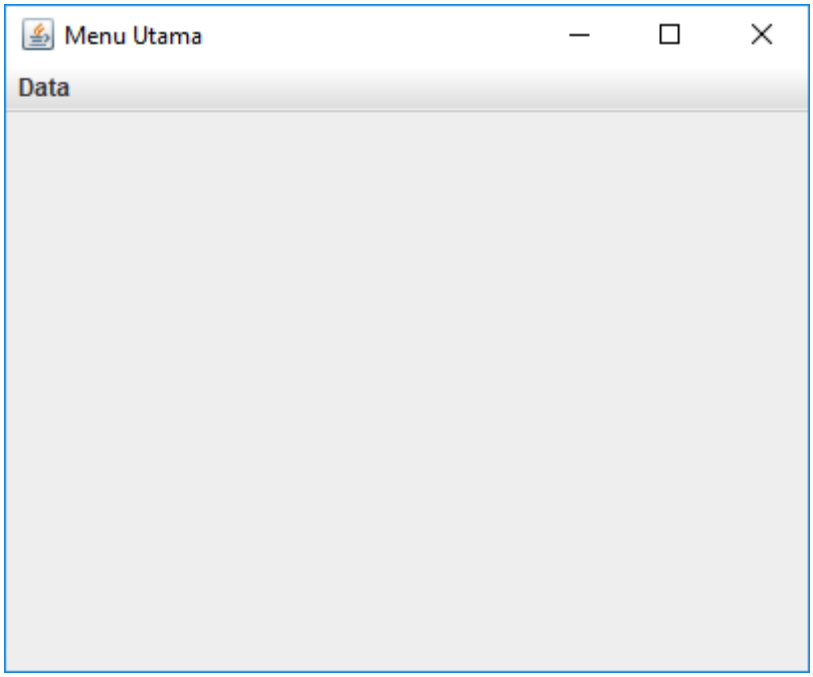

Projects
Files
Services
Databases
MySQL Server at localhost:3306 [root]
coba_1
Connect...
Delete
mysql
performance_schema
pesan_tiket
phpmyadmin
praktikum_8
si_stokbarang
test
Java DB
Drivers
jdbc:mysql://localhost:3306/praktikum_8



10. Selanjutnya tahapan uji coba program. Jalankan aplikasi tersebut.

    Halaman login.




Menu Utama
Data

DAFTAR PUSTAKA

[1] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java Virtual Machine Specification Java SE 8 Edition*. California: Addison-Wesley, 2014.

[2] K. S. Inc, *Java Server Programming Java EE 5*. New Delhi: Dreamtech Press, 2008.

[3] Y. D. Liang, *Introduction To Java Programming*, 9th ed. Pearson Education, 2016.

[4] D. Poo, D. Kiong, and S. Ashok, *Object-Oriented Programming and Java*, 2nd ed. London: Springer-Verlag London Limited, 2008.

[5] J. Dixit, *Computer Programming*, 2nd ed. New Delhi: Laxmi Publications.

[6] M. Somashekara, D. Guru, and K. Manjunatha, *OOP with Java*. New Delhi: PHI Learning Private Limited, 2017.

[7] B. J. Holmes and D. T. Joyce, *Object-Oriented Programming with Java*. Jones and Bartlett Publishers, 2001.

[8] M. Corporation, *Basic Java Programming for Kids and Beginners*. Bloomington: iUniverse, 2019.

[9] B. Teufel, *Organization of Pragramming Languages*. Springer-Verlag, 1991.

[10] B. Baesens, A. Backiel, and S. Vanden Broucke, *Beginning Java Programming The Object-Oriented Approach*. Indiana: John Wiley & Sons, 2015.

[11] P. Sengupta and B. B. Chaudhuri, *Object-Oriented Programming Fundamentals and Applications*. New Delhi: Prentice-Hall of India Private Limited, 2004.

[12] R. Liguori and P. Liguori, *Java Pocket Guide*. Sebastopol: O'Reilly Media, 2008.

[13] D. Reilly and M. Reilly, *Java Network Programming and Distributed Computing*. Boston: Pearson Education, 2005.

GLOSARIUM

A

ARRAY : variabel yang digunakan untuk menyimpan koleksi data yang bertipe data sama.

D

DECISION : struktur pemrograman untuk membuat percabangan pilihan dimana program akan dijalankan salah satunya, atau beberapa, jika kondisi terpenuhi.

E

Encapsulation : merupakan sebuah proses penggabungan atribut dan *method* ke dalam sebuah kelas.

G

Getter : method yang dibuat untuk mengambil atau mengembalikan nilai sebuah atribut kepada pemanggilnya.

I

Inheritance : memungkinkan sebuah kelas menurunkan variabel maupun *method* kepada kelas turunannya.

J

Java API : sebuah layer yang berisi kelas-kelas dan antarmuka pemrograman.

JDK : lingkungan pemrograman untuk kompilasi, *debubgging*, dan menjalankan aplikasi.

JRE : lingkungan perangkat lunak dimana program Java dijalankan.

JVM : mesin komputer yang mengeksekusi *bytecode* pada platform perangkat keras tertentu.

L

LOOPING : melakukan instruksi / perintah secara berulang selama kondisi terpenuhi (bernilai *true*).

M

Method : merupakan sekumpulan kode program yang diberikan nama.

P

Polimorfisme : kelas dapat mempunyai banyak bentuk *method* yang berbeda-beda meskipun namanya sama.

S

Setter : method yang dibuat untuk memodifikasi nilai dari suatu atribut berdasarkan nilai yang dikirim oleh pemanggilnya menggunakan parameter input.

INDEKS

PROFIL PENULIS

Noordin Asnawi adalah dosen Program Studi Sistem Informasi, Fakultas Teknik, Universitas PGRI Madiun. Alumni S1 STT Dharma Iswara Madiun dan S2 STMIK AMIKOM Yogyakarta. Buku ini merupakan buku ke 2 yang ditulis sejak tahun 2018 yang diterbitkan oleh UNIPMA Press.