PENGANTAR BAHASA PEMROGRAMAN JAVA

Johanes Eka Priyatma

Teknik Informatika

Fakultas Sains dan Teknologi

Universitas Sanata Dharma Yogyakarta

BAB I

Cara Kerja Komputer

1.1. Bahasa Mesin

Mesin komputer sangat kompleks dan terdiri atas banyak komponen yang saling berinteraksi satu sama lain. Dari komponen yang banyak itu, salah satu komponen terpentingnya adalah *CPU (Central Processing Unit)* yang bisa dianggap sebagai otaknya komputer. Unit inilah yang bertugas melakukan perhitungan serta menjalankan **program.**

Secara sederhana program adalah rangkaian (barisan) perintah yang **pasti dan tegas** yang harus dijalankan oleh komputer. Sedang komputer dibuat untuk menjalankan perintah yang ditulis dengan bahasa yang sangat sederhana yakni bahasa mesin. Setiap komputer mempunyai bahasa mesinnya sendiri. Artinya, satu jenis komputer dengan CPU yang berbeda pembuatnya belum tentu mampu menjalankan perintah bahasa mesin yang dikenal oleh komputer produk lain.

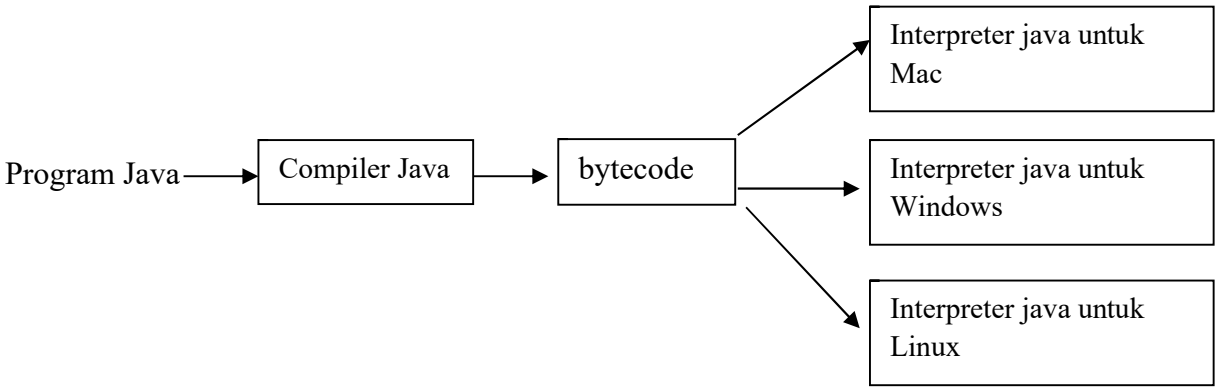
1.2. Mesin Java Virtual

Bahasa mesin merupakan rangkaian perintah sederhana yang dapat langsung dimengerti oleh CPU. Akan tetapi, menulis program dengan bahasa mesin sangat sulit dan merepotkan. Untuk itu dibuat bahasa yang lebih mudah ditulis dan dipahami manusia. Bahasa ini disebut dengan **bahasa tingkat tinggi**. Ada banyak bahasa tingkat tinggi seperti Pascal, C++, Java, Fortran, dll. Karena bahasa-bahasa ini tidak bisa langsung dimengerti oleh CPU komputer maka diperlukan penerjemah yang secara umum disebut dengan ***compiler*.** Penerjemah ini akan membaca perintah dalam bahasa tingkat tinggi kemudian mengubahnya ke dalam bahasa mesin tertentu, baru setelah itu komputer menjalankan perintah dalam bahasa mesin hasil terjemahan tersebut. Karena setiap jenis komputer hanya kenal dengan bahasa mesin tertentu maka hasil terjemahan *compiler* ini juga hanya bisa dijalankan oleh mesin tertentu tersebut.

Secara umum ada 2 pendekatan dalam menerjemahkan bahasa tingkat tinggi ke dalam bahasa mesin. Pendekatan pertama dilakukan dengan menerjemahkan dahulu semua perintah bahasa tingkat tinggi ke dalam bahasa mesin yang sesuai, baru kemudian hasilnya dijalankan oleh CPU. Ini yang dilakukan oleh *compiler*. Pendekatan kedua dilakukan dengan menerjemahkan

perintah bahasa tingkat tinggi satu per satu sesuai dengan alur perintah kemudian langsung dijalankan oleh CPU. Cara kedua ini seperti yang dilakukan oleh penerjemah lisan. Penerjemah yang memakai cara kedua ini disebut dengan ***interpreter***. Sedang cara pertama (compiler) melakukan penerjemahan seperti halnya penerjemah tulisan.

Perancang bahasa Java lebih memilih cara penerjemahan bahasa tingkat tinggi yang merupakan gabungan antara pendekatan *compiler* dan *interpreter*. Perancang Java tidak langsung menerjemahkan bahasa tingkat tinggi ke bahasa mesin tetapi ke bahasa antara yang disebut ***java bytecode.*** Bahasa *bycode* hasil terjemahan ini tergolong sederhana strukturnya tetapi belum bisa dijalankan oleh CPU karena setiap jenis komputer mempunyai bahasa mesin yang berbeda. Namun demikian tidaklah begitu rumit membuat interpreter yang mampu menerjemahkan *java bytecode* ke dalam bahasa mesin tertentu. Dengan pendekatan ini maka eksekusi program yang ditulis memakai Java dapat digambarkan sbb:



Gambar 1. Eksekusi Program Dalam Java

Dengan konsep seperti ini maka perancang Java membayangkan seolah-olah ada mesin imajiner/maya (hanya ada dalam bayangan) atau virtual yang bisa menjalankan *bytecode* tersebut. Komputer imajiner yang mampu menjalankan bytecode ini disebut dengan ***JVM/Java Virtual Machine*** **(Komputer maya Java).** Dengan konsep JVM ini maka bahasa Java seolah menjadi bahasa yang tidak tergantung kepada jenis prosessor komputer atau lebih dikenal dengan sebutan ***machine independent***.

1.3. Komponen Dasar Pemrograman

Ada dua komponen utama dalam sebuah program yakni **data** dan **perintah**. Untuk bisa mengelola *data* dengan baik maka Anda harus menguasai konsep *variabel* beserta tipe atau jenisnya. Sedang untuk menguasai perintah, Anda harus menguasai struktur pengendalian perintah dan metode atau fungsi. Anda akan menghabiskan banyak waktu untuk menguasai kedua konsep ini dengan baik.

Variabel adalah tempat untuk menyimpan data di memori komputer. Tempat ini diberi nama sehingga mudah dirujuk dan digunakan. Nama tempat di memori ini lalu disebut dengan nama variabel. Dalam praktek pembuatan program, Anda hanya berurusan dengan nama variabel saja sedang bagaimana tempat ini dikelola di memori sudah menjadi urusannya *compiler*. Dapat Anda bayangkan variabel sebagai sebuah kotak dengan label tertentu. Selanjutnya isi kotak ini dirujuk atau ditunjuk memakai nama labelnya. Nama label inilah yang juga disebut dengan nama variabel. Pemrogram yang bertugas membuat/memberi nama variabel ini sesuai dengan kepentingannya. Meskipun nama yang dibuat boleh sembarang sesuai dengan kepentingan pemrogram tetapi supaya benar dan mudah diterjemahkan oleh *compiler* maka pembuatan nama variabel harus mengikuti beberapa ketentuan.

Di Java dan di bahasa komputer yang lain, setiap variabel harus ditentukan jenisnya. Jenis varibel ini akan memengaruhi jenis data yang dapat disimpan di dalamnya. Dengan menetapkan jenis/tipe datanya maka *compiler* akan lebih efektif bekerja dan malah dapat membantu pemrogram untuk menulis program dengan konsisten dan benar. Ada banyak jenis data yang disimpan dan dikelola dengan cara yang berbeda oleh komputer. Ada data bilangan bulat (*integer*) misalnya 0, 20, 30, 5,… atau pecahan seperti 2.34, 56.78, 100.45, … atau data teks seperti “Bambang Samekti” dan “Tina Turner” ataupun data huruf seperti ‘A’, ‘B’, …,’E’ yang dipakai dalam nilai matakuliah. Masih ada **banyak jenis data lain** yang akan dibahas kemudian.

Bahasa pemrograman juga selalu mempunyai perintah untuk menyimpan atau mengambil data dari suatu variabel kemudian memakainya dalam perhitungan tertentu. Sebagai contoh, perintah *bunga = simpanan * 0.06;* berarti meminta komputer untuk mengalikan isi dari variabel *simpanan* dengan *0.06* kemudian hasilnya disimpan ke lokasi di memori dengan nama/alamat *bunga.* Perhatikan bahwa tanda ‘=’ (sama dengan) di sini bermakna lain dengan ‘=‘ di dalam rumus matematika karena di sini nilai ruas kiri tidak harus sama dengan nilai ruas kanan.

Tentu saja bahasa pemrograman juga mempunyai perintah untuk membaca data yang diberikan oleh *user*/pemakai melalui keyboard maupun file. Bahasa pemrograman juga mempunyai perintah untuk menampilkan data baik di layar komputer maupun di kertas lewat *printer*. Perintah dasar untuk membaca, melakukan perhitungan, serta menyimpan data dari dan ke memori merupakan komponen dasar dari sebuah program. Komponen dasar ini kemudian dirangkai secara sistemik memakai pola dan urutan perintah atau metode/fungsi tertentu yang akhirnya menjadi sebuah program komputer lengkap.

Secara singkat, program adalah barisan perintah. Perintah ini akan dikerjakan oleh komputer secara urut dari yang lebih dulu ditulis. Tetapi jika hanya memakai urutan begitu saja maka rangkain perintah ini akan sangat panjang. Untuk itu, bahasa komputer mempunyai fasilitas untuk mengendalikan pengerjaan perintah yang memungkinkan perintah tertentu dikerjakan atau dilewati dengan syarat tertentu. Bahasa pemrograman juga mempunyai perintah untuk mengulang sejumlah perintah tertentu sebanyak yang dikehendaki si pemrogram. Dua jenis pengendalian perintah ini masing-masing disebut dengan **percabangan** dan **perulangan**.

Sebagai contoh, jika pemrogram menghendaki bahwa perhitungan bunga tergantung kepada besarnya simpanan di mana jika simpanan lebih dari 10 juta maka bunganya 6 % tetapi jika kurang dari 10 juta bunganya hanya 5 % maka perintah menghitung bunga dapat ditulis menjadi

```
if (simpanan > 10000000)
        bunga = simpanan * 0.06;
else
        bunga = simpanan * 0.05;
```

Perhatikan bahwa perintah yang dipakai di atas memakai bahasa kata dari bahasa Inggris “if” dan “else” karena memang kosa kata bahasa Java berasal dari bahasa Inggris (sebab yang membuat juga berbahasa Inggris, meskipun ‘Java’ artinya ‘Jawa’. Mungkinkah membuat bahasa pemrograman memakai bahsa Indonesia atau bahkan bahasa Jawa ?). Namun demikian, nama variabel seperti “bunga” dan ”simpanan” dapat dibuat memakai bahasa apapun baik Indonesia, Jawa, Inggris bahkan bahasa entah berantah !

Sekarang andaikan Anda ingin menghitung bunga untuk semua nasabah suatu bank maka akan sangat tidak masuk akal kalau perintah yang sama di atas, Anda tulis sebanyak nasabah bank tersebut. Mengapa ? Karena itu berarti anda harus menuliskan perintah yang sama mungkin 1000 kali bahkan 1 juta kali. Untuk itu, bahasa pemrograman mempunyai perintah untuk mengerjakan rangkaian perintah secara berulang sesuai yang dikehendaki pemrogram. Perintah berikut akan mengulang penghitungan bunga sebanyak 1000 kali

```
for (i = 1; i <= 1000; i++) {
        if (simpanan[i]  > 10000000)
                bunga[i] = simpanan[i] * 0.06;
        else
                bunga[i] = simpanan[i] * 0.05;
}
```

di mana simpanan[i] dan bunga[i] masing-masing menyatakan bunga dan simpanan nasabah ke i.

Semakin kompleks masalah yang harus diselesaikan oleh si pemrogram akan menjadikan program semakin kompleks juga. Salah satu cara mengelola program yang kompleks adalah dengan memecahnya ke dalam sub-sub program yang biasa dikenal dengan nama ***subroutine/subprogram*** **atau metode atau fungsi** (meskipun secara detil masing-masing ini berbeda tetapi untuk sementara kita anggap sama). Misalkan penghitungan bunga tidak sesederhana di atas tetapi lebih rumit dan kompleks maka perhitungan tersebut bisa Anda pisahkan ke dalam subprogram dengan nama hitungBunga() maka perhitungan bunga untuk 1000 nasabah menjadi lebih sederhana karena secara konseptual yang Anda lakukan cukup memanggil subprogram hitungBunga() sebanyak 1000 kali, memakai perintah berikut:

```
for (i = 1; i <= 1000; i++) {
        nasabah[i].hitungBunga();
}
```

Memakai konsep subprogram ini, program yang kompleks bisa disusun menjadi lebih sistematis dan sederhana. Bayangkan seperti Anda membuat atau merakit mobil-mobilan Tamiya tetapi komponen-komponen sudah ada dan Anda tinggal merangkainya. Komponen seperti roda, body, dynamo, dll bisa dipandang sebagai sebuah subprogram. Bedanya, dalam Tamiya Anda

tinggal membeli komponen tersebut tetapi ketika Anda membuat program Java Anda harus mengambil dari internet atau membuatnya sendiri!!!!

1.4. Mengapa Bahasa Java ?

Di awal dikatakan bahwa ada banyak tersedia bahasa tingkat tinggi selain Java. Mengapa yang kita pelajari bahasa Java ? Jawaban pertanyaan ini tidak ada hubungannya dengan letak Universitas Sanata Dharma (USD) di pulau Jawa yang sebagian besar mahasiswanya berbahasa Jawa. Bahasa Java dapat dikatakan sebagai bahasa pemrograman yang paling banyak digunakan di dunia. Tidaklah mengherankan bila bahasa Java adalah bahasa yang paling banyak diajarkan di seluruh perguruan tinggi di dunia. Java merupakan pilihan terbaik untuk pembuatan program yang bisa dijalankan di berbagai jenis komputer. Oleh karena itu, bahasa Java sangat sesuai untuk pembuatan program di internet karena di sana terhubung berbagai jenis merk komputer.

Kehebatan bahasa Java juga didukung oleh cara Java memodelkan realitas. Sebelumnya, yakni di era 80an, kompleksitas pemrograman komputer diatasai dengan pendekatan **terstruktur**. Dalam pendekatan ini program yang kompleks dipecah ke dalam bagian-bagian yang lebih kecil, dan bagain yang lebih kecil dipecah lagi ke dalam bagian-bagian yag lebih kecil lagi, demikian seterusnya. Setiap bagian kecil bisa dianggap sebagai sebuah modul. Namun demikian, pendekatan terstruktur ini terpisah dari data yang merupakan komponen terpenting lain dari program. Selain hal tersebut, ternyata program yang sudah tersusun tidak begitu mudah untuk digunakan kembali (*reused*).

Kenyataan ini mendorong lahirnya cara baru dalam menyusun program yang mengintegrasikan antara proses dan data. Program akan dilihat seperti halnya realitas sehari-hari sebagai kumpulan obyek seperti manusia, mobil, hewan, dll. Setiap obyek mempunyai data dan tingkah lakunya sendiri. Obyek-obyek tersebut saling berinteraksi dengan mengirim pesan satu ke yang lain. Pemrogram yang disusun dengan menirukan realitas ini disebut dengan **PBO/ pemrograman berorientasi obyek (*obyek oriented programming*)**. Java dirancang untuk bisa digunakan dengan konsisten memakai gagasan PBO ini. Dengan memakai pendekatan PBO, terbukti program yang ditulis memakai Java lebih handal, mudah dikembangkan serta bisa dijalankan di berbagai jenis komputer. Selamat datang di pemrograman Java !!!

1.5. Praktikum Pengenalan Lingkungan Pengembangan Program Java

a. Fase *Edit*

Fase 1 merupakan fase penyuntingan (*edit*) file program Java menggunakan program *editor*. Dalam fase ini, program Java (yang sering juga disebut *source code*) diketik menggunakan *editor*, dilakukan koreksi-koreksi sejauh diperlukan, dan selanjutnya disimpan dalam media simpanan sekunder, seperti hard disk maupun flash disk. *Source code* Java tersebut diberi nama dengan ekstensi **.java** yang menunjukkan bahwa file tersebut berisi *source code* Java. Salah satu program editor dalam lingkungan sistem operasi Windows yang dapat digunakan untuk membuat program Java adalah **notepad**.

Untuk mengembangkan suatu sistem informasi bagi organisasi, tersedia pula ***integrated development environment (IDE)***, yang diproduksi oleh beberapa *software supplier*. IDE menyediakan alat bantu (*tools*) yang mendukung proses pengembangan perangkat lunak, termasuk ***editor*** untuk mengetik dan mengedit program serta ***debugger*** untuk menemukan *error* dalam program.

Beberapa IDE yang populer diantaranya adalah NetBeans (www.netbeans.org), jEdit (www.jedit.org), Eclipse (www.eclipse.org), JBuilder (www.borland.com), JCreator (www.jcreator.com), BlueJ (www.blueJ.org) dan jGRASP (www.jgrasp.org).

b. Fase *Compile*

Dalam fase 2, *programmer* menggunakan perintah **javac** (*Java compiler*) untuk mengkompilasi suatu program. Sebagai contoh, untuk mengkompilasi program Welcome.java , kita perlu mengetikkan perintah berikut dalam window terminal sistem yang dipakai (misal *command prompt*)

```
javac Welcome.java
```

Jika program telah selesai dikompilasi, maka *Java Compiler* akan menghasilkan file bertipe **.class** yang berisi program versi kompilasi**.** Dalam contoh kita, maka akan dihasilkan file Welcome.class. *Java Compiler* mengubah *Java source code* menjadi **bytecodes** yang berisi

perintah-perintah yang akan dijalankan selama fase eksekusi (*execute*). Bytecodes akan dieksekusi oleh **Java Virtual Machine (JVM)** yang merupakan bagian dari JDK (Java Development Kit) dan dasar dari *java platform.* Sebuah *Virtual Machine* (VM) merupakan sebuah perangkat lunak aplikasi yang mensimulasikan sebuah komputer tetapi menyembunyikan detail sistem operasi dan perangkat keras dari program yang berinteraksi dengan VM. Jika VM yang sama digunakan pada banyak komputer dengan platform yang berbeda, maka aplikasi tersebut dapat digunakan pada semua platform. Misal program Java yang telah kita kompilasi pada sistem operasi Windows, kita jalankan pada komputer dengan sistem operasi Linux dan menggunakan JVM, maka program tetap akan berjalan dengan baik.

Sebagai contoh, untuk mengeksekusi aplikasi Java yang telah kita kompilasi (Welcome.class) kita tinggal mengetikkan perintah pada terminal sistem (command prompt)

java Welcome

kemudian JVM akan mulai bekerja memulai fase 3.

a. Fase *Loading* Program ke dalam Memori

Dalam fase 3 ini, program akan ditempatkan dalam memori sebelum proses *loading* dimulai. *Class loader* akan mengambil file .class yang berisi *bytecodes* dari program dan mengirimkan ke memori. Disamping file .class dari program kita, *class loader* juga akan memanggil file .class dari Java yang digunakan oleh program kita. File .class tersebut dapat diambil dari sistem kita atau dari jaringan / internet.

b.* Fase *Bytecode Verification

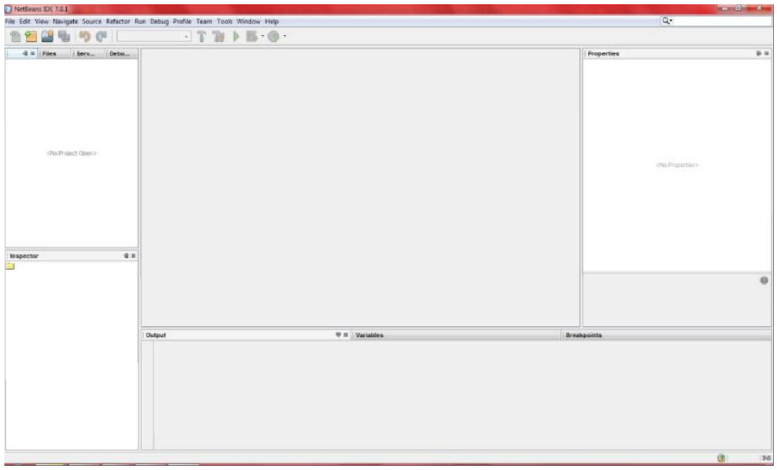
Dalam fase 4, setelah kelas di-*load, bytecode verifier* akan menguji *bytecodes* untuk meyakinkan bahwa *bytecodes* benar dan tidak melanggar batas-batas keamanan Java.

e. Fase *Execution*

Dalam fase 5, JVM akan mengeksekusi *bytecodes* program, kemudian melakukan perintah seperti yang diberikan program. JVM akan melakukan proses translasi *bytecodes* ke dalam bahasa mesin yang bersesuaian dengan platform komputer yang digunakan untuk mengeksekusi.

f. Pengenalan IDE NetBeans

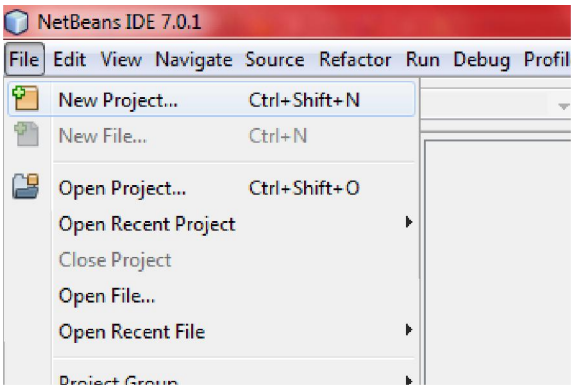
NetBeans merupakan sebuah IDE (integrated development environment) yang menyediakan fitur-fitur untuk memudahkan programmer dalam memprogram Java. NetBeans dapat didownload secara free di situs http://www.netbeans.org. Dengan NetBeans, maka proses penulisan source code, kompilasi, eksekusi, debugging (mencari kesalahan program) menjadi sangat mudah. Gambar 1.1 memperlihatkan tampilan awal dari NetBeans yang berisi berbagai layar.



Gambar 1.1. Lingkungan NetBeans

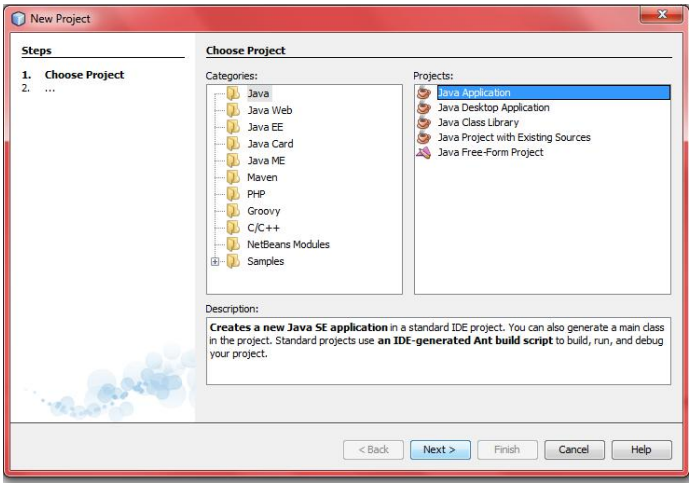
Dalam NetBeans, semua langkah pemrograman dilakukan dalam sebuah project. Sebuah project akan berisi source code dan semua lingkungan pemrograman yang diperlukan untuk mengkompilasi dan mengeksekusi source code. Sebuah project dapat berisi lebih dari satu source code program. Berikut ini adalah langkah-langkah untuk memulai project :

1. Buka NetBeans IDE melalui Start -> NetBeans -> NetBeans IDE
2. Setelah IDE NetBeans terbuka, pilihlah File -> New Project (Ctrl-Shift-N) seperti tampak pada gambar 1.2.



Gambar 1.2. Memilih menu Project Baru

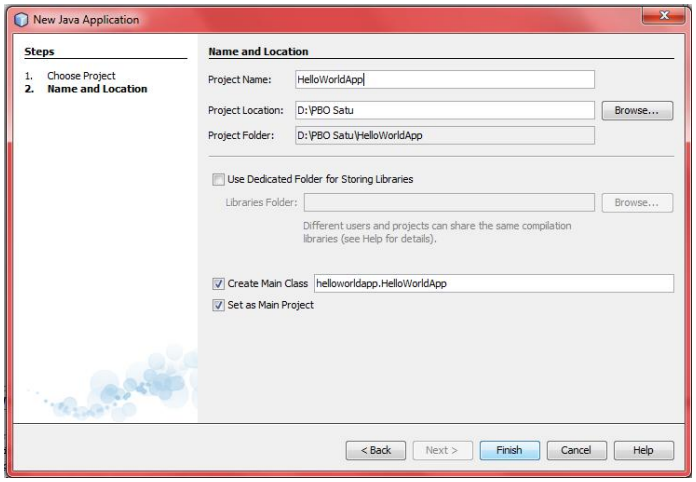
3. Berikutnya, pada jendela New Project, pilihlah Categories Java -> Projects Java Application, seperti tampak pada gambar 1.3. Setelah itu klik Next.



Gambar 1.3. Membuat Java Application Baru

4. Pada jendela New Java Application, pilihlah
    a. Project Name : HelloWorldApp.
    b. Project Location : (nama folder tempat anda akan menyimpan project).
    c. Pilihlah Create Main Class seperti tampak pada gambar 1.4
    d. Pilihlah Set as Main Project.

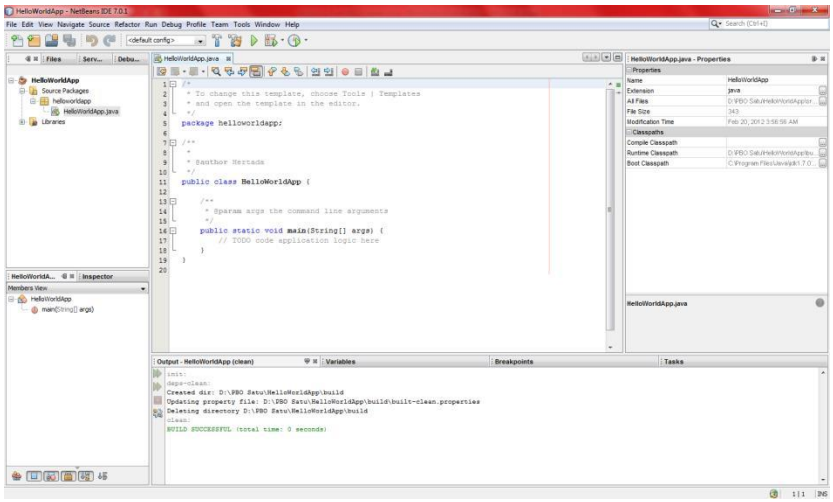
e. Klik Finish.



Gambar 1.4. Membuat Java Application Baru

Akan terbentuk sebuah project pada IDE. Komponen-komponen pada layar IDE seperti tampak pada gambar 1.5. adalah sebagai berikut :

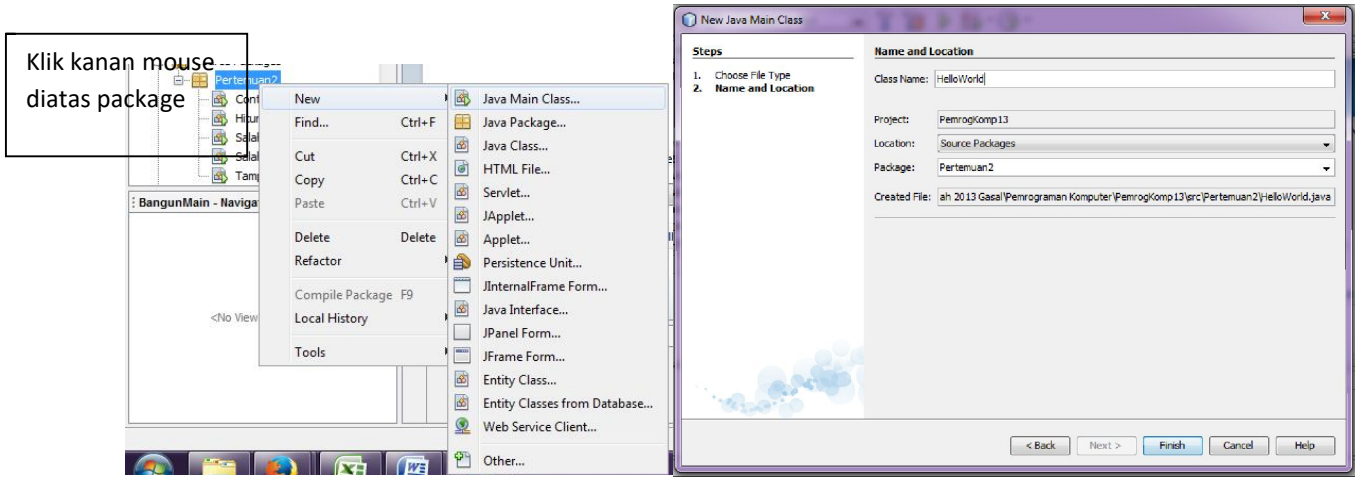
- Layar Project, berisi tampilan bentuk pohon yang menggambarkan komponen dari sebuah project berupa file berisi source code, banyak library yang digunakan oleh kode program kita.
- Layar Editor, berisi kode program dari program kelas HelloWorldApp kita.
- Layar Navigator, yang digunakan untuk mengarahkan secara cepat setiap elemen pada kelas yang kita pilih.
- Layar Output, yang berisi keluaran program yang dijalankan, atau keterangan program yang dicetak.
- Layar Properties, berisi atribut/data, method dan keterangan dari kelas yang kita buat.



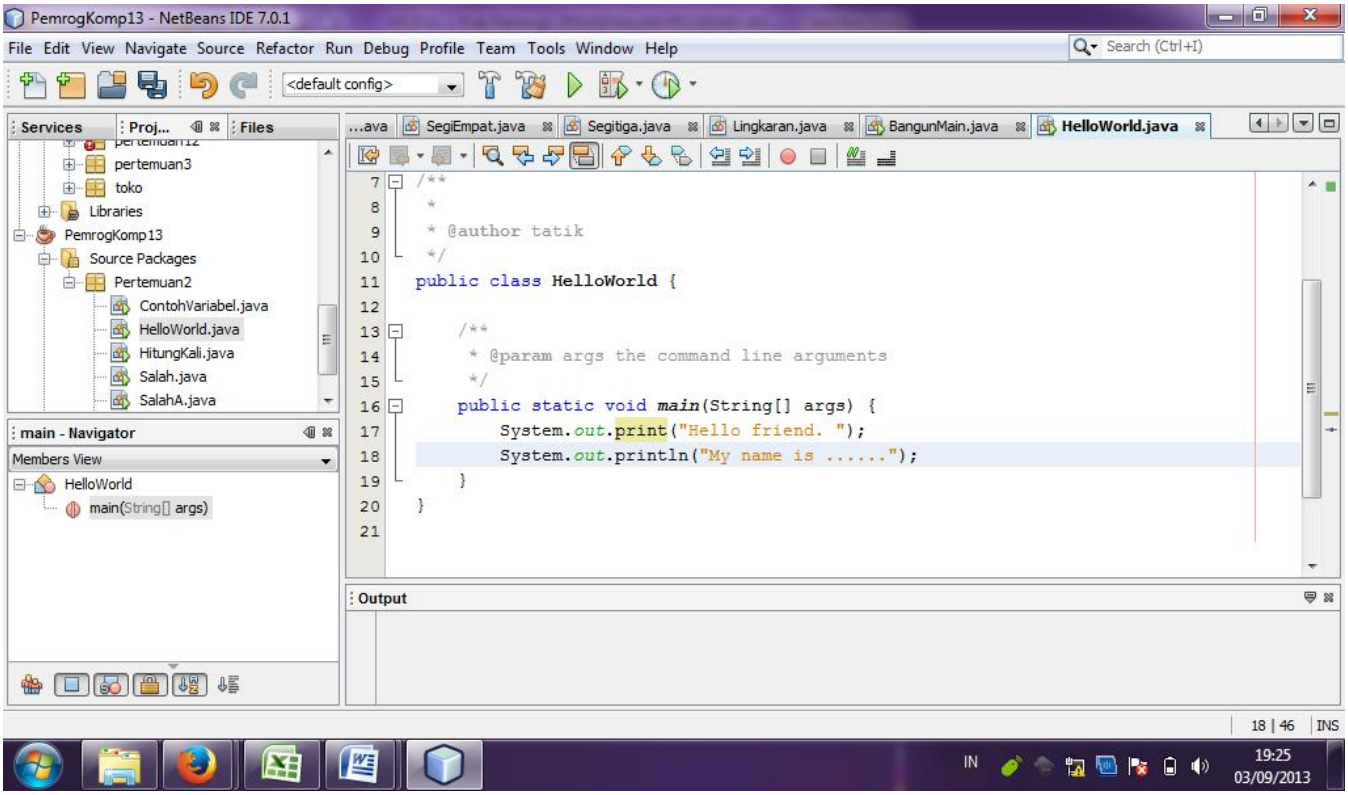
Gambar 1.5. Membuat Java Application Baru

Setelah project terbentuk, tampak pada jendela project sebuah kelas **main** yang bernama HelloWorldApp. Kelas main merupakan sebuah kelas yang berisi method main (public static void main(String[] args). Jika sebuah kelas di-run, maka eksekusi akan dimulai dari baris pertama method main. Jika sebuah kelas tidak memiliki method main, maka kelas tersebut tidak dapat di-run. Method-method dalam kelas yang tidak memiliki method main akan dijalankan dari method main atau dari method lain.

Untuk lebih jelasnya, kita buat kelas baru bernama HelloWorldApp.java (gambar 1.6), dan kita isikan ke dalam kelas tersebut perintah-perintah seperti pada gambar 1.7

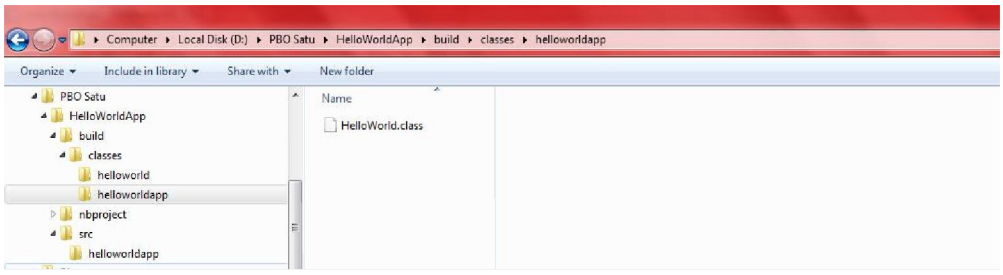


Gambar 1.6. Membuat kelas baru.



Gambar 1.7. Membuat kelas HelloWorld.java.

Simpanlah file HelloWorld.java dengan memilih File->Save atau Ctrl-S. Dan kini kita sudah memiliki sebuah kelas bernama HelloWorld.java. Proses Save pada NetBeans diset otomatis untuk melakukan proses Compile terhadap file yang kita simpan, sehingga kita tidak perlu melakukan fase compile secara terpisah. Tampak pada gambar 1.8, isi dari folder project kita setelah proses Save file HelloWorld.

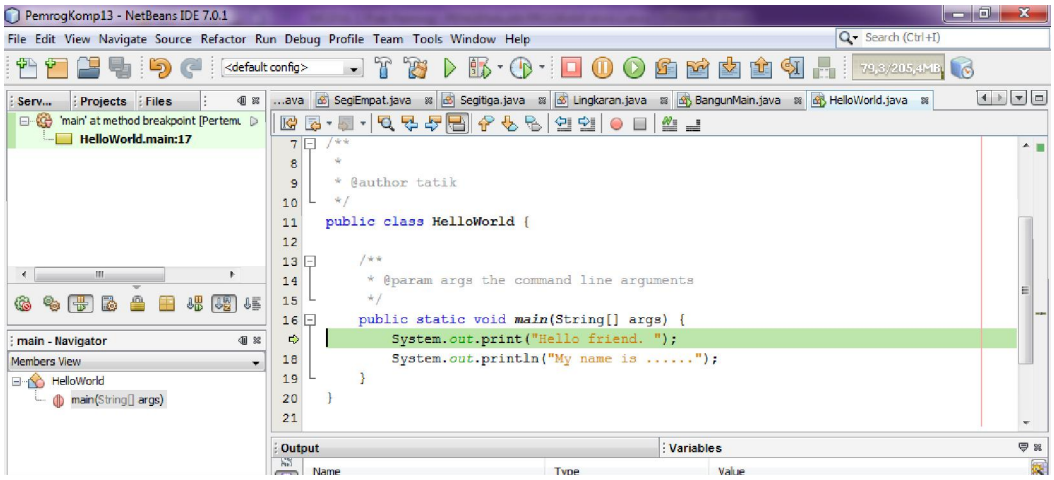


Gambar 1.8. Isi folder project.

Simpan HelloWorld.java dan kemudian kita lakukan proses run program dengan cara pilih Run->Run Main Project (F6). Karena pada project hanya terdapat satu kelas main, yaitu HelloWorld.java, maka otomatis method main yang dijalankan adalah method main pada kelas HelloWorld.java. Sehingga hasil dari run program tersebut adalah tampak pada gambar 1.9.

Gambar 1.9. Output program HelloWorldApp.java.

Untuk mengetahui jalannya program, kita dapat melakukan proses debug program dengan menggunakan kunci F7 dan F8. Posisi perintah dimana program sedang dijalankan, tampak dari arsiran pada layar, seperti tampak pada cuplikan debug gambar 1.10.



Gambar 1.10. Cuplikan debug program HelloWorldApp.java.