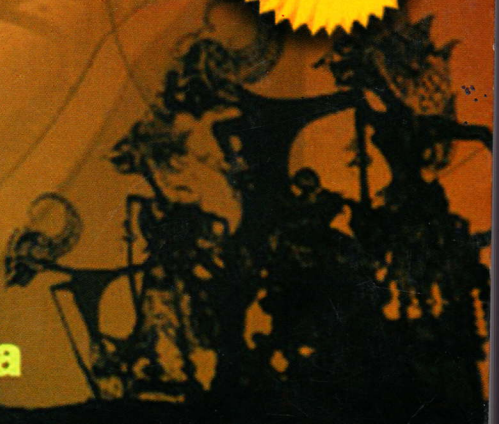
Tuntunan Pemrograman JAVA

Jilid 1

Edisi Revisi



STIKOM

Rangsang Purnama

TUNTUNAN PEMROGRAMAN JAVA

Jilid 1

EDISI REVISI

TUNTUNAN PEMROGRAMAN JAVA

Jilid 1

EDISI REVISI

Rangsang Purnama

PRESTASI PUSTAKA
PUBLISHER

PRESTASI PUSTAKA
PUBLISHER
Penerbit, Jakarta

Tuntunan Pemrograman Java, jilid 1, edisi Revisi

Penulis : **Rangsang Purnama**
Desain Cover :**Abdullah Syafik Noer**
Setting : **Tim Prestasi Pustaka**

Perpustakaan Nasional : Katalog Dalam Terbitan (KDT)

Prestasi Pustaka

Tuntunan Pemrograman Java, jilid 1, edisi Revisi
14cm x 20.5cm

ISBN 978-979-9433-89-3

Cetakan Pertama Edisi Revisi
Desember 2007

KATA PENGANTAR

Beberapa minggu setelah serial "Tuntunan Pemrograman Java" terbit, kami menerima banyak masukan dari para pembaca dan pengamat. Ada yang mengusulkan agar buku tersebut disisipi CD yang berisi kode program dan sistem Java, ada yang mengkritik kode program di dalam buku yang ternyata ada *error*-nya, dan lain-lain. Terima kasih kepada pembaca yang telah memberi kritik, saran dan masukan.

Buku yang sedang Anda baca ini merupakan edisi revisi dari buku sebelumnya. Kami telah melakukan beberapa revisi dan pembenahan agar materi dan metode penyampaiannya menjadi lebih baik lagi. Kode-kode program yang ada dalam setiap pokok bahasan telah diperbaiki dan diuji sebelum dimasukkan ke dalam buku. Yang juga tidak kalah penting adalah kami telah menambahkan soal-soal latihan yang relevan dengan topik bahasan di akhir bab.

Buku jilid 1 memuat topik-topik dasar sebagai pengantar ke pemrograman Java yang lebih tinggi. Buku ini terdiri dari 6 bab. Bab I berisi tutorial singkat bagaimana caranya menginstall sistem Java dan EditPlus. Kegagalan di dalam proses instalasi sistem Java akan menyebabkan kegagalan di dalam pembuatan program Java, karena kita tidak dapat melakukan pengujian terhadap program yang kita buat.

Bab 2 merupakan pengantar untuk mempelajari bahasa Java. Beberapa langkah untuk mulai membuat program Java diuraikan di sini. Selain itu juga dijelaskan pengertian variabel dan tipe data yang berlaku di Java. Bagi yang suka dengan tampilan ala Windows, pada materi ini dijelaskan bagaimana kita bisa mengatur tampilan untuk output data pada area window.

Bab 3 berisi penjelasan tentang struktur kontrol percabangan. Seperti pada bahasa pemrograman yang lain, Java juga mengenal instruksi "if" yang digunakan untuk proses percabangan. Di sini diuraikan bagaimana *style* percabangan di

Java. Penulisan program dibuat sedemikian rupa agar kode program mudah dibaca.

Bab 4 berisi penjelasan tentang struktur kontrol perulangan. Pembahasan dimulai dari struktur "for" termasuk modifikasinya, lalu instruksi "while" dan akhirnya instruksi "do-while". Bagi Anda yang pernah menggunakan C atau C++ tentu tidak asing dengan instruksi ini.

Bab 5 berisi penjelasan tentang variabel array. Di sini dijelaskan tentang array satu dan dua dimensi. Keunggulan dari penggunaan variabel array juga dijelaskan, misalnya mencari data dari sekumpulan data dan melakukan proses pengurutan data. Salah satu implementasi array dua dimensi adalah matrik. Pada bab ini dijelaskan pula operasi penjumlahan dan perkalian matrik. Kemudian pada akhir bab dijelaskan tentang tiga class standar dari Java yang mendukung pemahaman tentang variabel array, yaitu Vector, Stack dan Hashtable.

Bab 6 berisi penjelasan tentang sub program. Secara umum, sub program terbagi menjadi dua yaitu prosedur dan fungsi. Bab ini menjelaskan tentang sub program yang dibuat tanpa parameter dan sub program yang dibuat dengan parameter. Sebagai tambahan, Java mengenal konsep *function-overloading*. Di akhir bab dijelaskan tentang fungsi rekursif.

Akhir kata, kritikan dan saran dapat dikirim melalui e-mail pada alamat rangsang@stikom.edu.

Surabaya, Oktober 2007

Penulis

DAFTAR ISI

BAB – 1
INSTALASI SISTEM

Tujuan

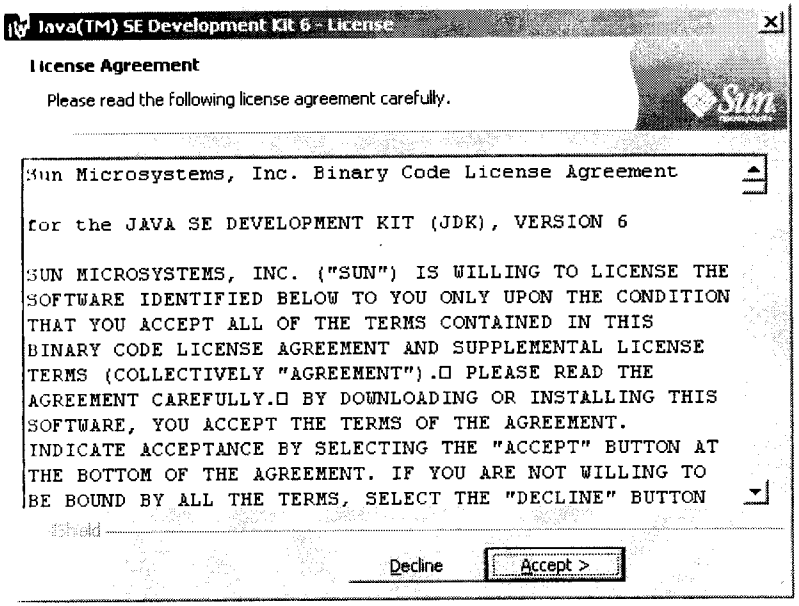
1. Pembaca mengetahui persyaratan sistem komputer untuk bisa menjalankan sistem Java dan aplikasinya.
2. Pembaca mampu menginstalasi sistem Java.
3. Pembaca mampu menginstalasi EditPlus dan bisa melakukan setting EditPlus terhadap fungsi-fungsi Java.

1.1 Instalasi Java

Langkah pertama yang harus dilakukan agar dapat mengkompilasi dan menjalankan program Java adalah menginstall sistem Java. File sistem Java bisa diperoleh secara cuma-cuma melalui internet pada alamat http://java.sun.com/javase/downloads/index.jsp. Beberapa syarat penting yang harus diketahui sebelum menginstall sistem Java adalah:

1. Sistem operasi yang digunakan harus mendukung penamaan file lebih dari 8 karakter dan penamaan ekstensi file lebih dari 3 karakter. Contoh sistem operasi yang bisa digunakan adalah Windows 2000 dan sesudahnya, Linux, atau Solaris.
2. Harus tersedia ruang kosong pada hard disk minimal 700 MB untuk instalasi yang membutuhkan fasilitas Documentation.
3. Pada komputer harus terpasang mouse untuk memudahkan pemilihan topik instalasi maupun instruksi pada fasilitas Documentation.
4. Processor yang digunakan adalah jenis Pentium II atau yang lebih baik.

Proses instalasi dimulai dengan menjalankan file jdk-6-windows-i586.exe yang merupakan sistem Java versi JDK 1.6.0. JDK singkatan dari Java Development Kit. Instalasi dimulai dengan membaca file sumber Java. Setelah melalui beberapa proses singkat, instalasi Java akan menampilkan form persetujuan.

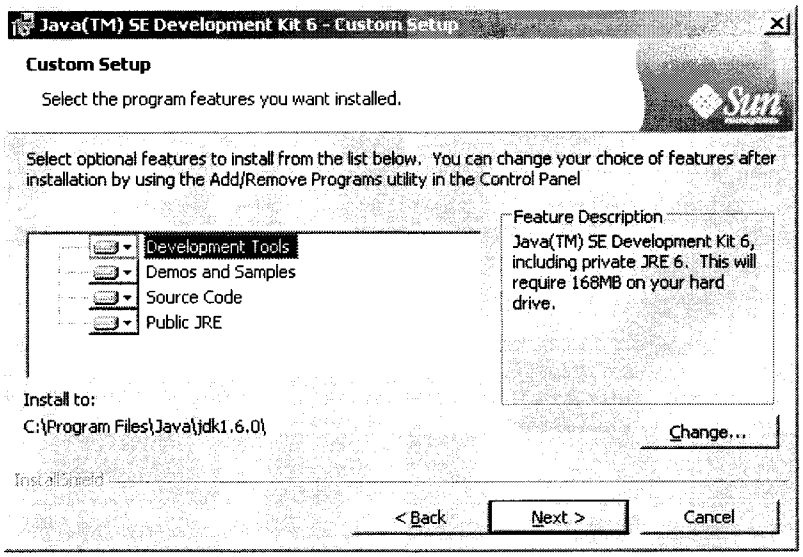


Klik "Accept >" untuk melanjutkan proses instalasi.

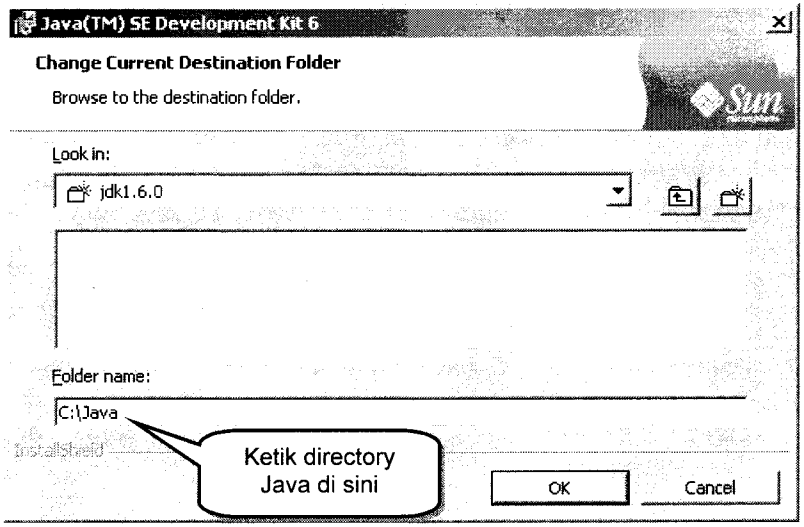
Langkah selanjutnya adalah menentukan lokasi di mana kita akan meletakkan sistem Java. Pada awalnya JDK 1.6.0 akan menawarkan directory "c:\Program Files\Java\jdk1.6.0" sebagai lokasi *default*. Kami sarankan agar Anda mengganti directory ini dengan nama lain. Beberapa pertimbangan:

1. Penggunaan karakter *spasi* pada nama directory bisa menimbulkan masalah pada beberapa aplikasi yang berkepentingan dengan Java.
2. Penamaan "jdk1.6.0" pada nama directory boleh jadi akan meninggalkan jejak pada *registry* sistem operasi ketika kita menghapus (*uninstall*) Java untuk beberapa keperluan, misalnya mengganti versi Java dengan yang baru.

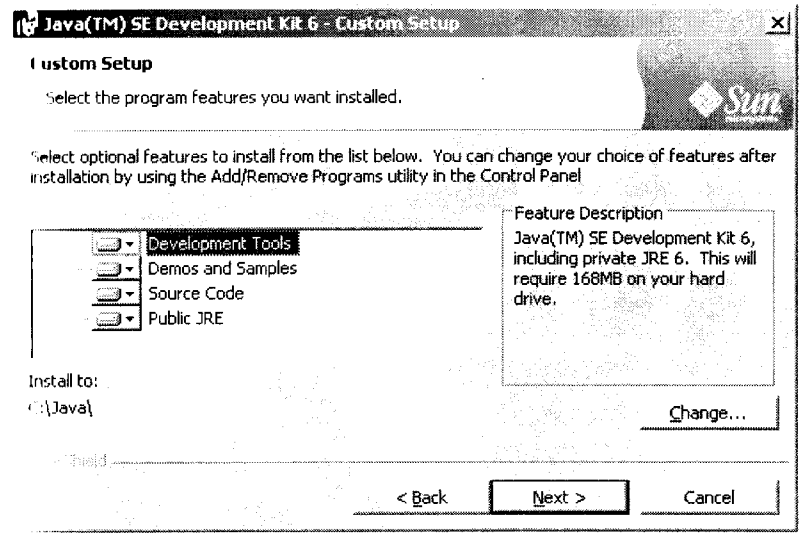
Kami sarankan untuk meng-*install* Java pada directory "c:\Java".



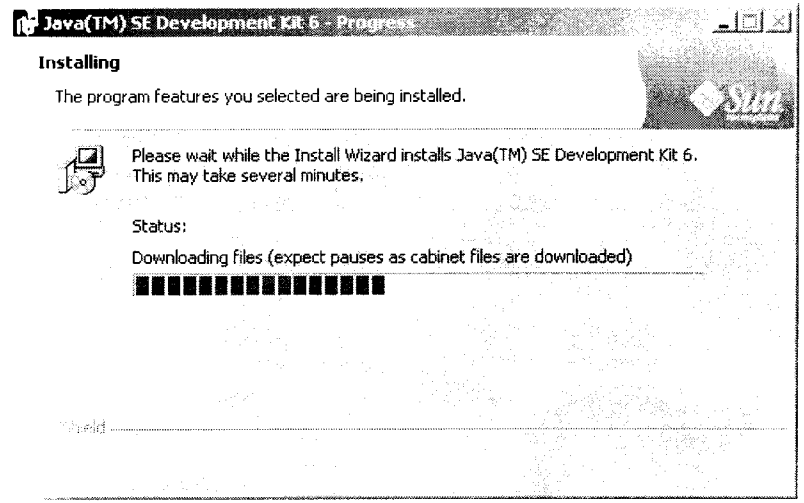
Klik "Change..." untuk mengganti directory Java.



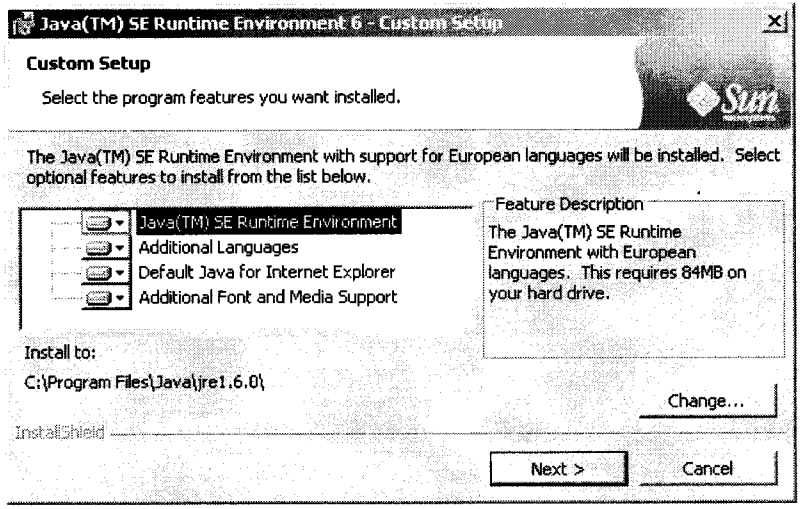
Klik "OK" untuk mengaktifkan directory target.



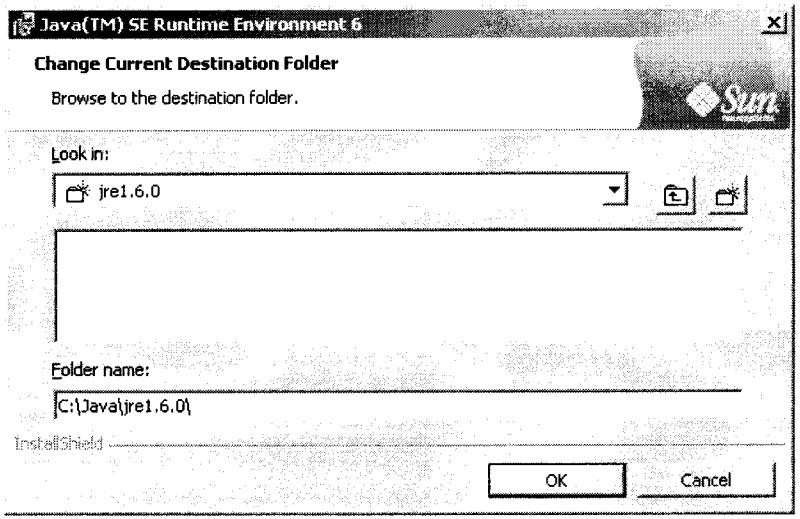
Klik "Next >" untuk memulai proses instalasi.



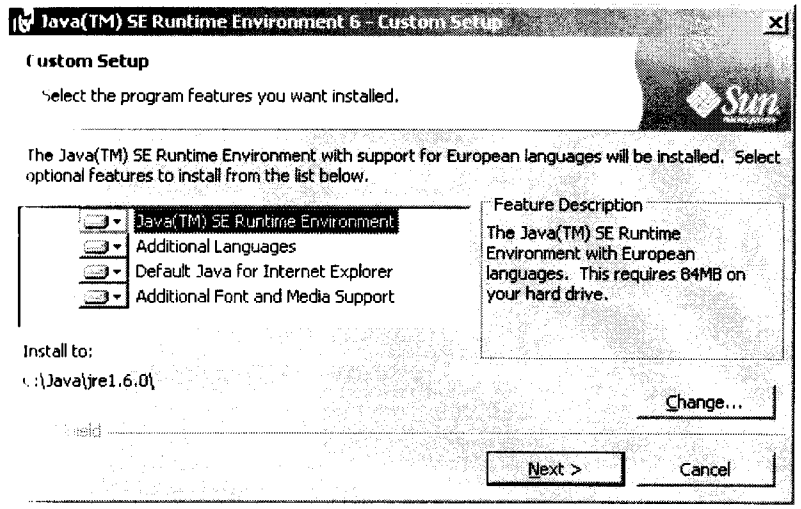
Tahap berikutnya adalah menentukan directory untuk Java Runtime Environment (JRE).



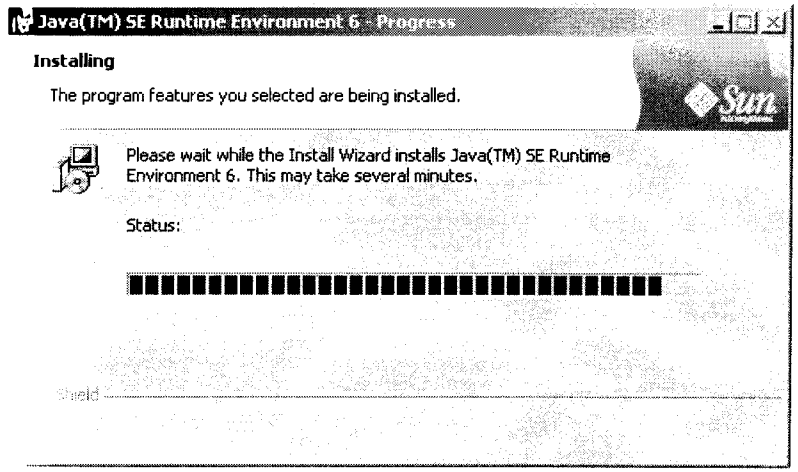
Klik "Change..." untuk mengganti directory yang ditawarkan sesuai dengan directory Java.



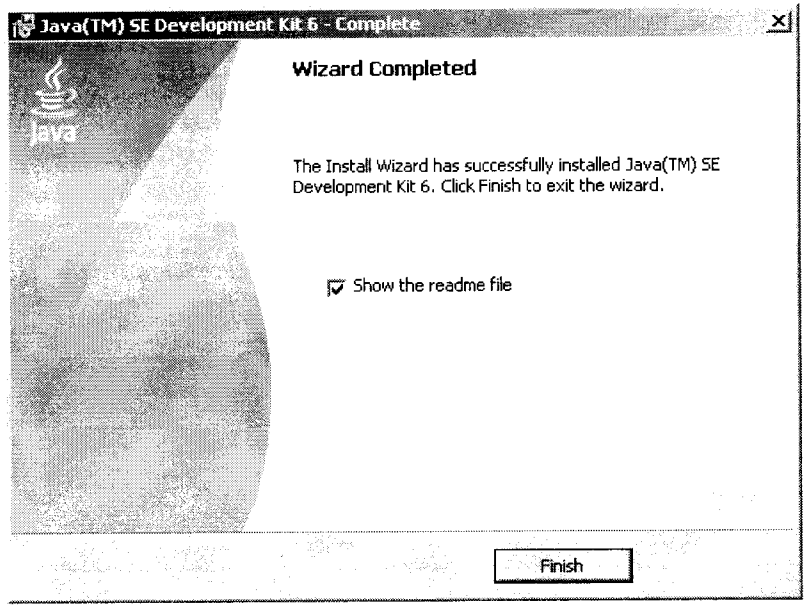
Klik "OK" untuk mengaktifkan directory target.



Klik "Next >" untuk melanjutkan proses instalasi.



Setelah proses instalasi selesai, Java akan menampilkan form penutup.



Directory "c:\Java\bin" berisi file-file yang dibutuhkan untuk mengolah file program Java. Untuk tingkat yang paling dasar, kita hanya membutuhkan 2 (dua) file:

1. File "javac.exe" yang digunakan untuk meng-*compile* file program Java.
2. File "java.exe" yang digunakan untuk meng-*execute* file program Java yang telah di-*compile*.

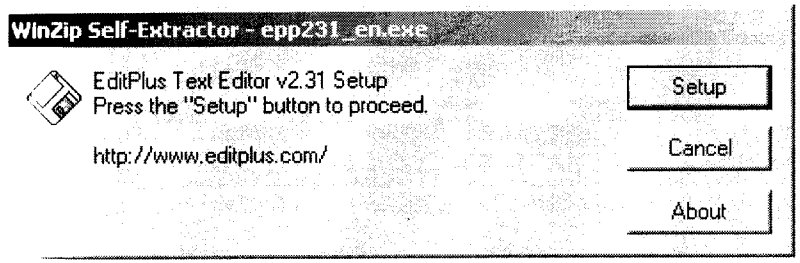
1.2 Instalasi EditPlus

Kehadiran Java tidak disertai dengan sarana untuk membuat atau mengedit kode program Java. Kita bisa melakukan pengetikan kode Java menggunakan aplikasi NotePad atau text editor lain

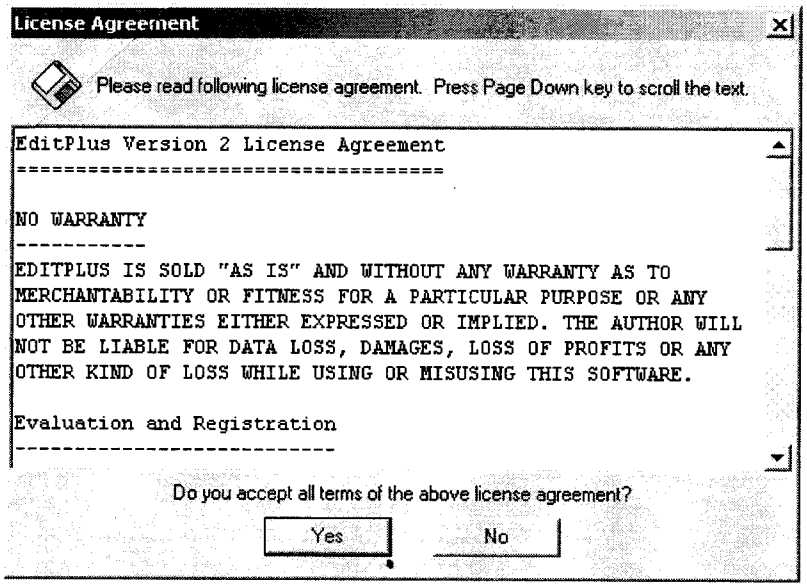
yang bisa menghasilkan naskah text. Konsekuensinya adalah kita harus sering berpindah program aplikasi: (1) mengetik program, (2) meng-*compile* dan *running*. Untuk menghindari kerepotan itu diperlukan adanya software lain yang didesain untuk membantu pengetikan naskah program, EditPlus salah satunya.

EditPlus merupakan produk dari ES-Computing. EditPlus merupakan software komersial yang hanya bisa digunakan selama 30 hari sebagai masa percobaan. EditPlus bisa diperoleh melalui web site http://www.editplus.com. EditPlus adalah sebuah aplikasi lepas yang dibuat untuk memudahkan user mengetik naskah program dari sembarang bahasa pemrograman yang telah di-*setting* sebelumnya. Pada kondisi awal, EditPlus bisa digunakan untuk mengetik program Perl, C/C++, dan Java. Dengan teknik tertentu EditPlus bisa digunakan untuk mengetik program Pascal, Assembler, dan lain-lain. EditPlus menawarkan banyak fitur untuk menambah kenyamanan dalam mengetik naskah program; dan itu terlalu luas untuk dibahas di sini.

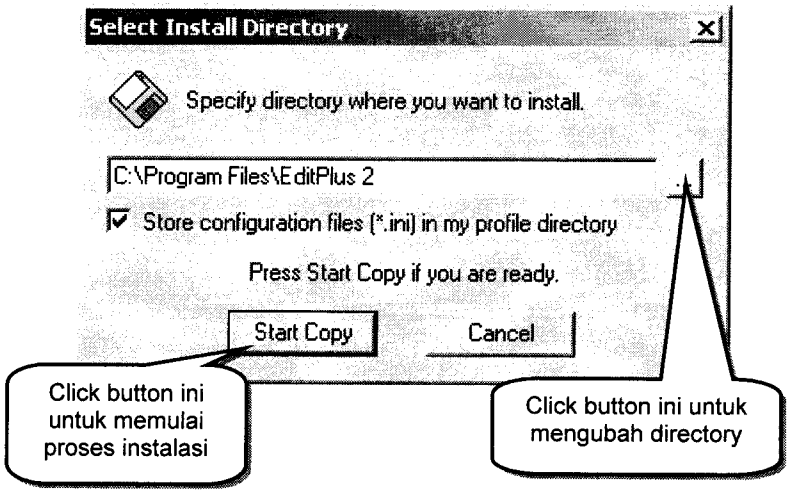
Instalasi EditPlus dimulai dengan menjalankan file "epp231_en.exe". Di sini kita menggunakan EditPlus versi 2.31.



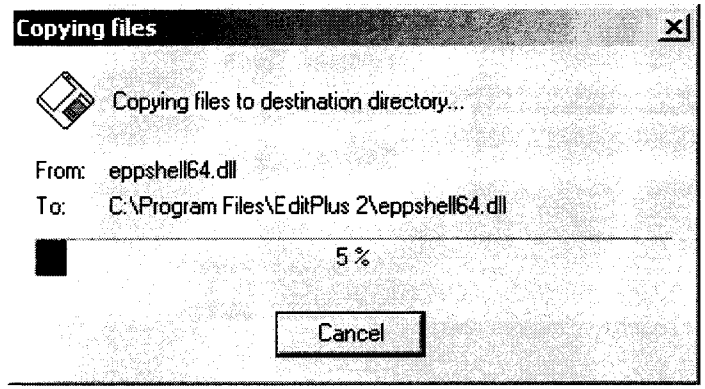
Click button "Setup", selanjutnya komputer akan melakukan proses extract file-file yang dibutuhkan untuk proses instalasi. Seperti kebanyakan program yang akan diinstall di suatu sistem komputer, EditPlus akan menampilkan License Agreement yang harus disetujui oleh pengguna jika ingin menggunakan program ini.



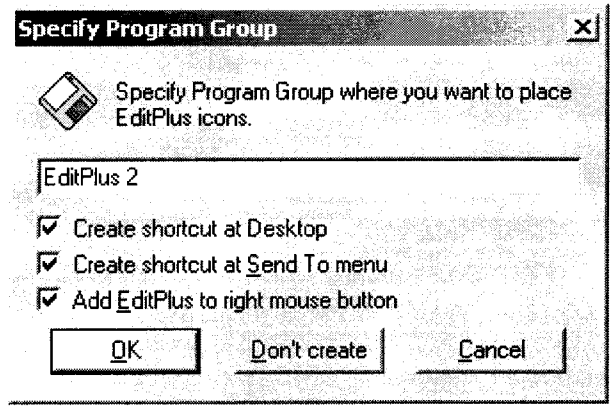
Langkah selanjutnya adalah menentukan lokasi di directory mana kita ingin EditPlus diletakkan, baru kemudian proses instalasi akan dilakukan sampai selesai.



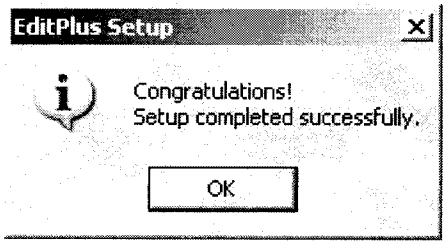
Klik "Start Copy" untuk memulai proses instalasi. Kami tidak menyarankan Anda untuk mengubah directory EditPlus.



Setelah proses instalasi selesai, sistem akan bertanya mengenai nama group bagi EditPlus.



Klik "OK" untuk mengakhiri proses instalasi. Kami tidak menyarankan Anda untuk mengubah nama group bagi EditPlus.



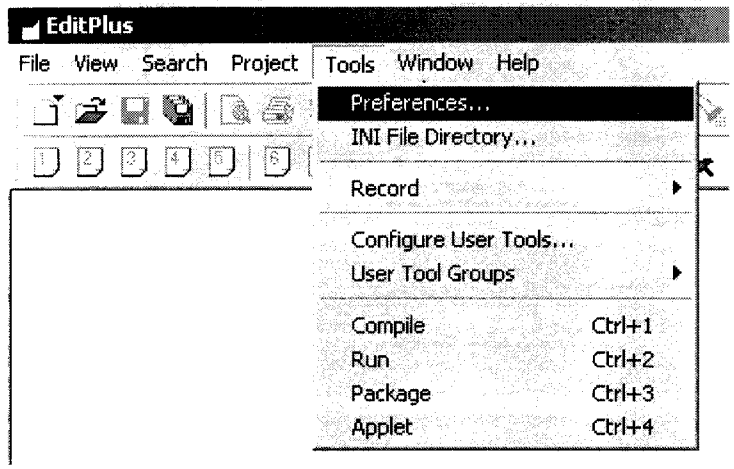
Aplikasi EditPlus siap digunakan.

1.3 Setting EditPlus

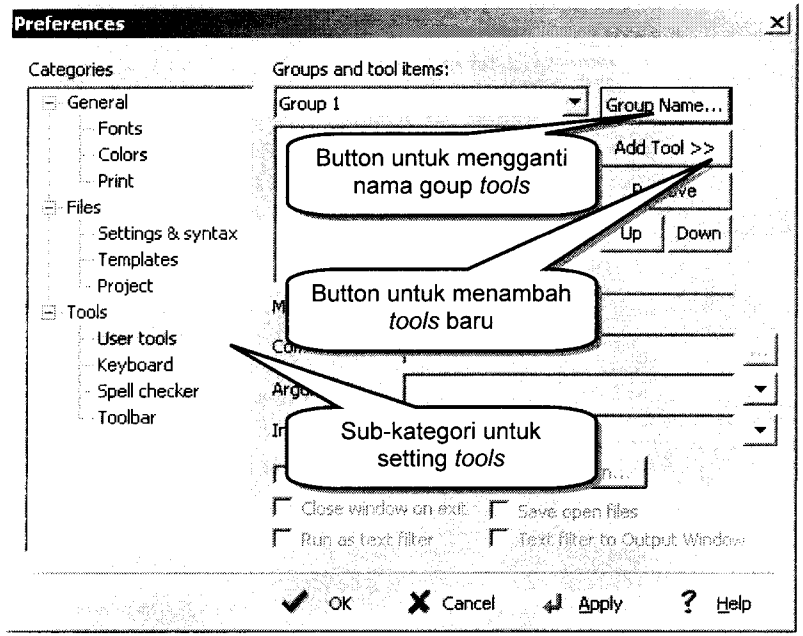
Pada dasarnya penggunaan EditPlus tidak terikat kepada bahasa pemrograman tertentu, sehingga EditPlus tidak dilengkapi dengan fitur untuk melakukan kompilasi dan eksekusi program. EditPlus dapat digunakan untuk melakukan pengetikan file-file berjenis text, HTML, Java, C/C++ dan Perl. Selain itu user juga diberi kebebasan untuk membuat *template* bagi file bertipe selain yang disediakan. Meski demikian, EditPlus telah dilengkapi dengan kemampuan untuk melakukan proses *compile* dan *running* aplikasi yang dibuat menggunakan bahasa pemrograman tertentu.

Jika Anda ingin meng-*compile* dan menjalankan program Java melalui DOS-command, Anda harus melakukan beberapa langkah yang membosankan; dan ini yang kita hindari. Berikut ini akan dijelaskan bagaimana kita bisa men-*setting* EditPlus agar bisa digunakan untuk membuat, meng-*compile* dan menjalankan aplikasi Java.

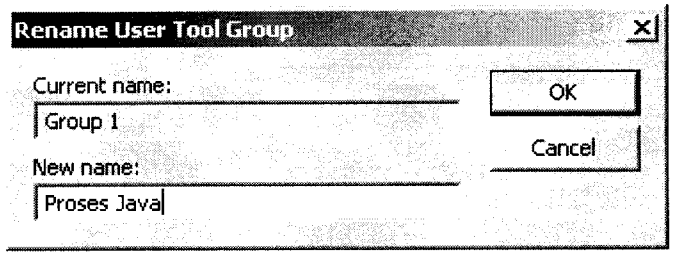
1. Jalankan program EditPlus, aktifkan menu "Tools" lalu pilih sub menu "Preferences...".



2. Pada form "Preferences", carilah kategori "Tools" dan sub-kategori "User tools". Aktifkan sub-kategori ini.

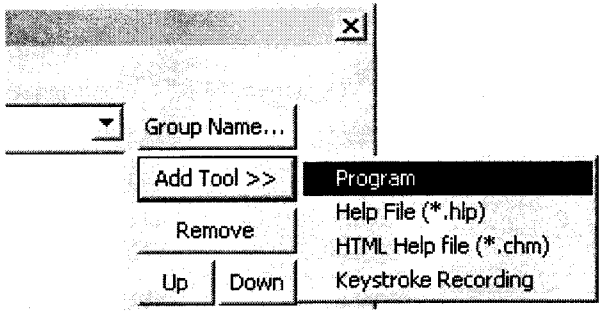


3. Klik "Group Name..." untuk mengganti nama group. Secara *default* nama group adalah 'Group 1', 'Group 2' dan seterusnya sampai 'Group 10'.

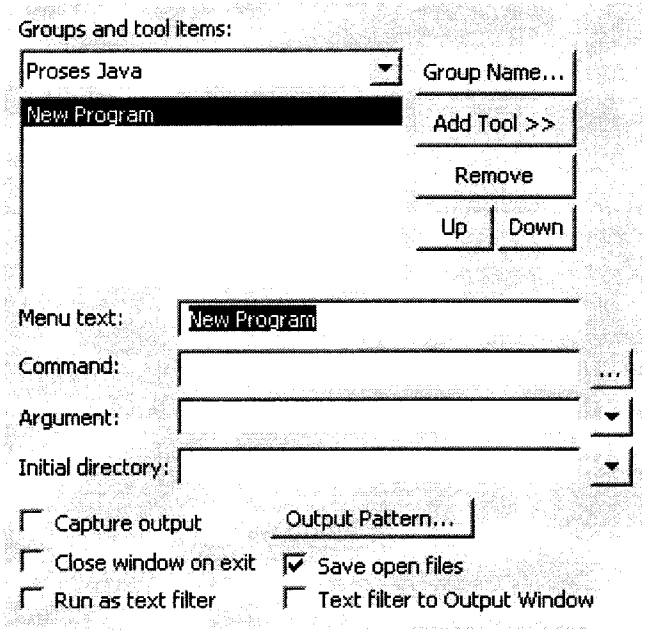


Klik "OK" untuk mengaktifkan nama group tersebut.

4. Klik "Add Tool >>" untuk menambah *tools* baru.



Pilih sub menu "Program" untuk membuat *tools* berupa *shortcut* dari sebuah file bertipe *application*.

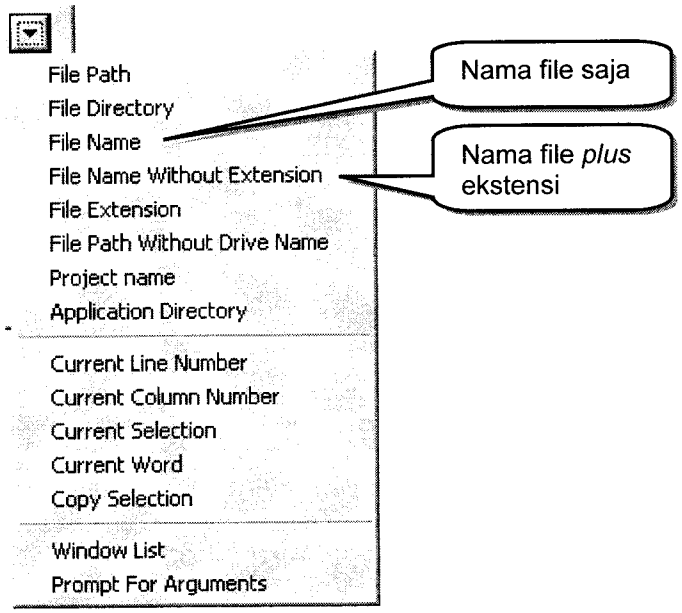


Tampilan ini akan selalu Anda temui setiap kali Anda membuat *shortcut* baru.

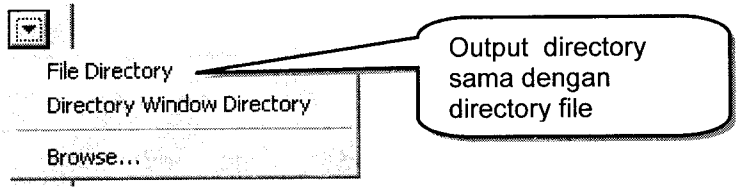
- Area "Menu text" diisi dengan sembarang teks yang mendeskripsikan judul *tools*.

- Area "Command" diisi dengan nama file *application* yang akan dieksekusi. Nama file ini juga bisa dipilih melalui tombol [...] yang terletak di sebelah kanan area ini.

- Area "Argument" diisi dengan *argumen* atau *parameter* yang diperlukan oleh program tersebut. Daftar *parameter* yang tersedia bisa dilihat dan dipilih dengan cara klik tombol [▼] yang terletak di sebelah kanan area ini.



- Area "Initial directory" diisi dengan nama directory yang menjadi lokasi eksekusi program. Daftar *parameter* yang tersedia bisa dilihat dan dipilih dengan cara klik tombol ▼ yang terletak di sebelah kanan area ini.



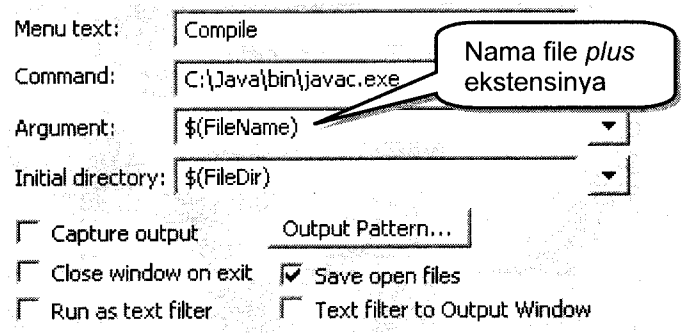
- Checkbox "Capture output" digunakan untuk mengatur apakah output dari "Command" akan ditampilkan pada layar DOS atau pada layar khusus EditPlus.

- Checkbox "Close window on exit" digunakan untuk mengatur apakah output berupa layar DOS akan segera

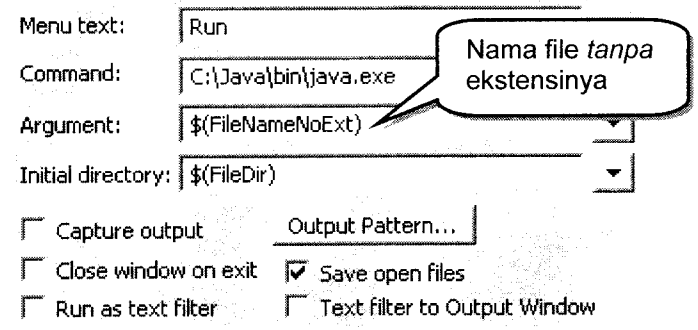
ditutup segera setelah aplikasi berakhir atau akan ditunda sampai user menekan tombol tertentu.

- Checkbox "Save open files" digunakan untuk mengatur apakah file-file yang sedang terbuka akan disimpan terlebih dulu sebelum "Command" dijalankan atau tidak.

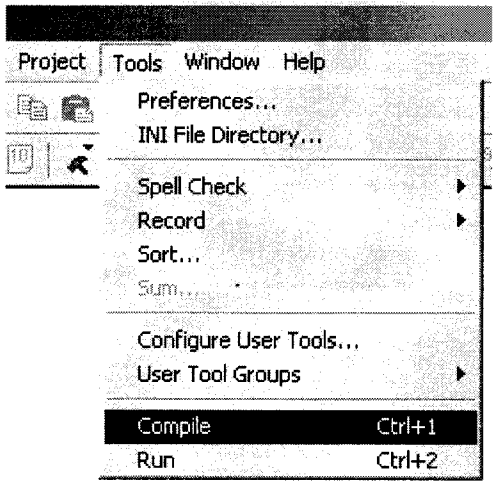
5. Untuk melakukan link dengan file "javac.exe" untuk tujuan meng-*compile* program Java, isikan data-data berikut ini: (diasumsikan bahwa lokasi sistem Java adalah c:\Java)



6. Untuk melakukan link dengan file "java.exe" untuk tujuan *running* program Java, isikan data-data berikut ini: (diasumsikan bahwa lokasi sistem Java adalah c:\Java)



Jika sekarang Anda perhatikan menu "Tools" dari EditPlus maka pada bagian akhir menu ada tambahan daftar *shortcut* yang telah kita definisikan tadi.

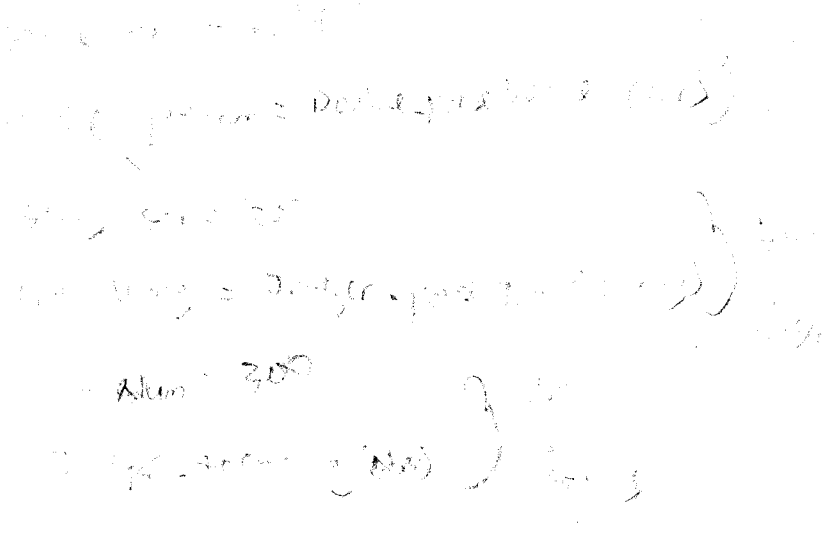


Proses kompilasi program Java juga dapat dilakukan dengan menekan kombinasi tombol 'Ctrl+1'; sedangkan untuk menjalankan program Java dapat digunakan kombinasi penekanan tombol 'Ctrl+2'. Sekarang EditPlus sudah siap digunakan sebagai editor bagi program Java.

BAB – 2
PENGANTAR BAHASA JAVA

Tujuan

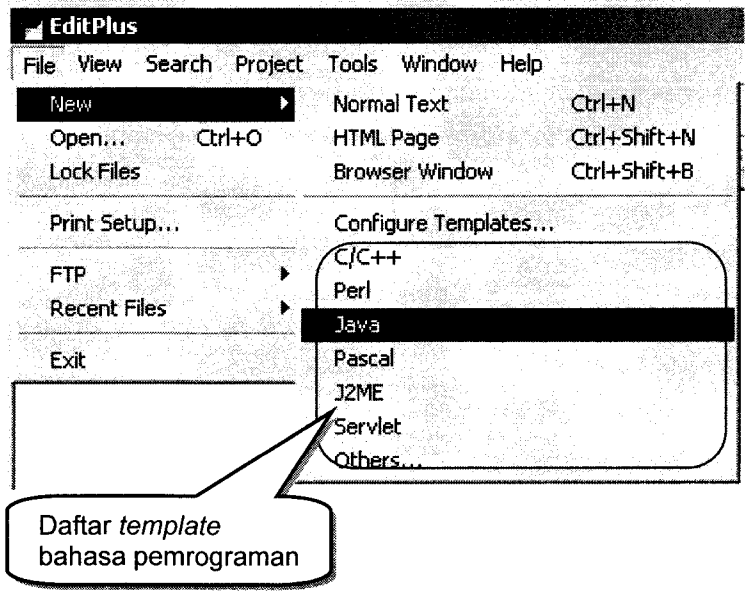
1. Pembaca mampu membuat aplikasi Java sederhana.
2. Pembaca mampu membuat aplikasi Java yang berisi proses input / output data.
3. Pembaca mampu melibatkan class JOptionPane dalam program.
4. Pembaca mampu menggunakan operator bawaan Java.
5. Pembaca mampu menangkap kesalahan proses (*exception*).
6. Pembaca mampu melibatkan konstanta dan komentar.



2.1 Tahap Persiapan

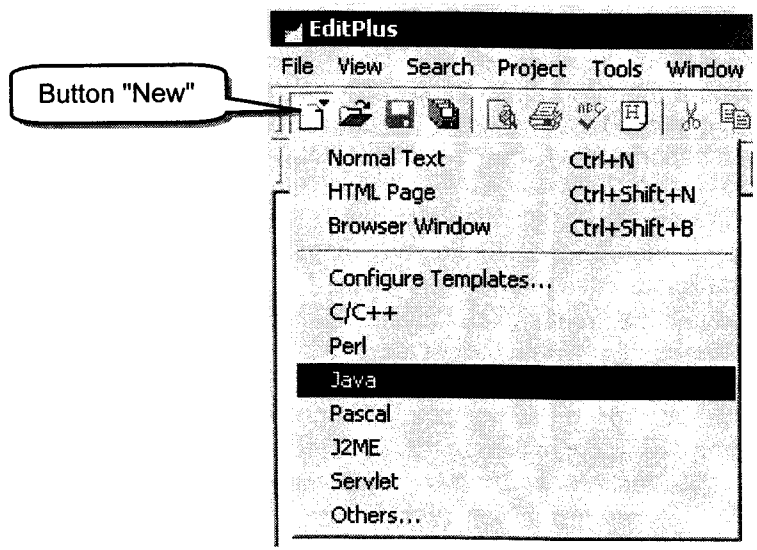
Hal pertama yang perlu diketahui adalah bagaimana membuat aplikasi Java menggunakan EditPlus. Anda bisa menggunakan satu dari dua cara berikut ini:

1. Aktifkan menu EditPlus dengan urutan berikut: "File > New > Java".

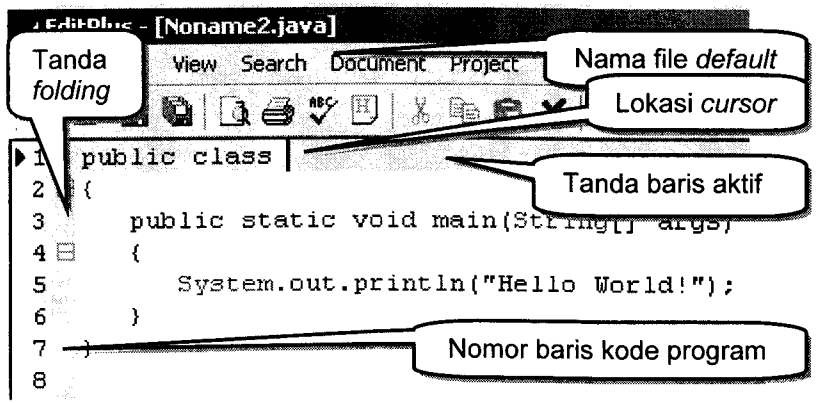


Pada tampilan di atas terlihat ada banyak *template* bahasa pemrograman yang di-*support* oleh EditPlus. Sebenarnya secara *default* EditPlus hanya mendukung tiga *template*, yaitu C/C++, Perl dan Java. Daftar *template* di atas adalah hasil modifikasi yang kami lakukan sesuai dengan kebutuhan di komputer kami.

2. Klik button "New" yang ada pada *toolbar* standar.



Apa pun cara yang Anda gunakan, selanjutnya EditPlus akan menampilkan sebuah naskah standar Java –demikian juga untuk bahasa pemrograman lainnya. Ini salah satu kelebihan EditPlus.

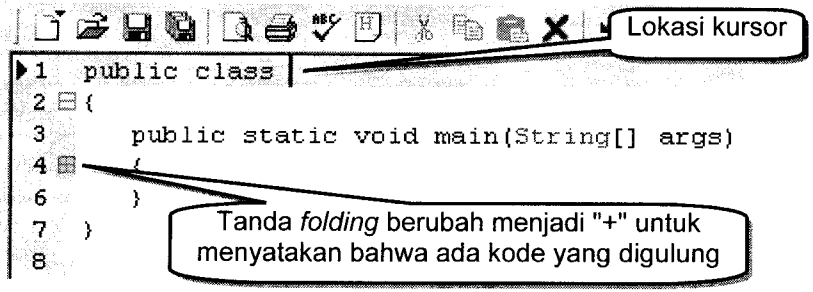


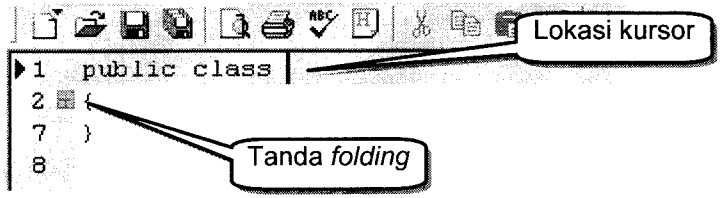
Setelah *template* ditampilkan, lokasi kursor langsung diletakkan di lokasi yang memerlukan pengetikan user. Sebuah program Java terbentuk dari sejumlah *class*, sehingga lokasi kursor akan diarahkan di belakang kata kunci "class", tujuannya agar user segera mengisi bagian ini sebagai suatu keharusan dari program Java. Anda bisa mencoba fasilitas ini untuk membuat file bertipe lain seperti C/C++, Perl maupun HTML.

Hal lain yang menjadi kelebihan EditPlus adalah penggunaan warna pada beberapa kata yang memiliki makna khusus, misalnya kata kunci Java. Konsep ini disebut *syntax-highlight*. Metode ini sangat menguntungkan dalam hal memudahkan pembacaan program karena setiap identitas ditulis dengan warna tertentu.

EditPlus 2.31 hadir dengan fitur baru berupa *code-folding*, yaitu sarana yang digunakan untuk "menggulung" naskah program yang berada dalam satu blok sehingga hanya tampak satu-dua baris yang mewakilinya. Dengan demikian programmer tidak perlu pusing melihat kode program yang tampil panjang lebar memenuhi layar.

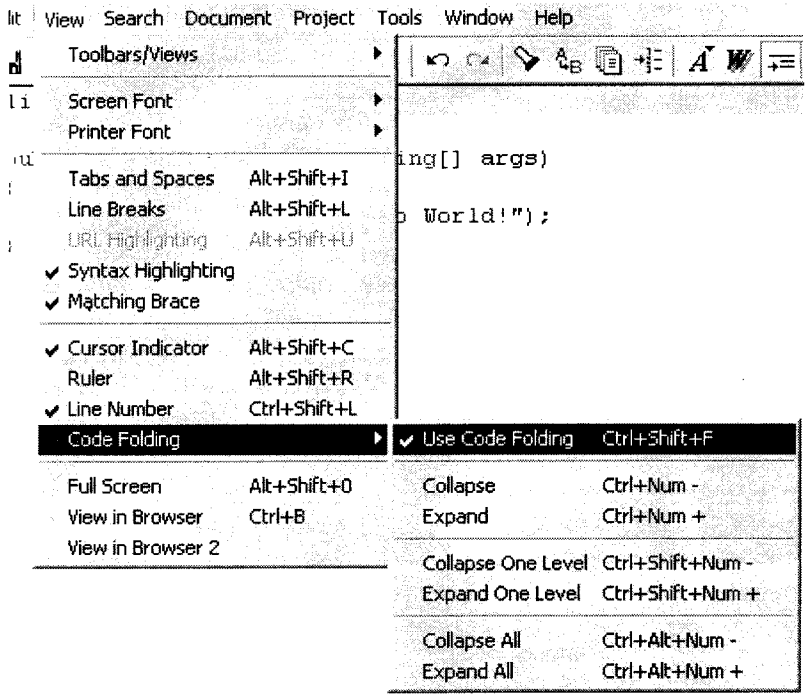
Berikut ini ditunjukkan bagaimana fitur *code-folding* mampu menggulung kode Java.





Pada saat fitur *code-folding* sedang beraksi, nomor urut kode program akan ditampilkan berlompatan karena ada nomor baris yang ikut tergulung.

Fitur *code-folding* diaktifkan melalui menu "View > Code Folding > Use Code Folding" atau melalui penekanan kombinasi tombol keyboard "Ctrl-Shift-F".



2.2 Membuat Program Java Pertama

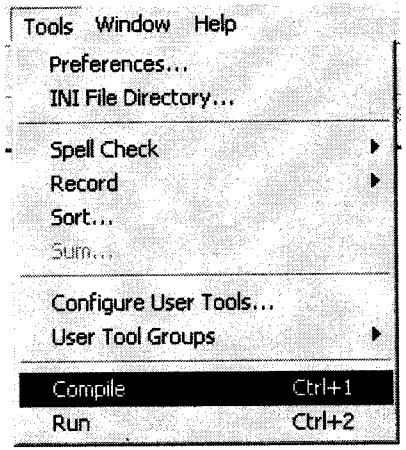
Sebagai langkah awal membuat program Java, ketiklah program singkat berikut ini: (nomor baris hanya untuk referensi)

Program 2.1

```
1  public class Program21
2  {
3    public static void main(String[] args)
4    {
5      System.out.println("Hello, World!");
6    }
7  }
```

Simpan file tersebut dengan nama "Program21.java". Sebuah program Java disimpan ke dalam file dengan nama yang sama dengan nama class-nya, tepat sama baik huruf kapital maupun huruf kecilnya. Ekstensi file haruslah ".java".

Sekarang kita akan meng-*compile* file tersebut. Aktifkan menu "Tools" lalu pilih *item* "Compile" atau nama apa pun yang telah Anda tulis ketika membuat tools ini.

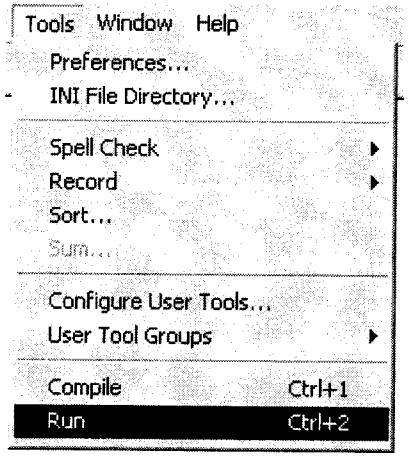


Hasil kompilasi program adalah:

```
C:\WINDOWS\system32\cmd.exe
Press any key to continue . . .
```

Pesan ini menyatakan bahwa program berhasil di-*compile* tanpa ada kesalahan. File hasil *compile* adalah "Program1.class". Setiap file hasil *compile* akan memiliki ekstensi .class.

Langkah berikutnya adalah menjalankan program tersebut. Aktifkan menu "Tools" lalu pilih *item* "Run" atau nama apa pun yang telah Anda tulis ketika membuat tools ini.



Output program:

```
C:\WINDOWS\system32\cmd.exe
Hello, World!
Press any key to continue . . .
```

Kata "Press any key to continue. . ." adalah pesan dari EditPlus, bukan dari Java. Pesan ini menyatakan bahwa program sudah berakhir, dan kendali proses dikembalikan ke EditPlus.

Java menyediakan dua perintah untuk menampilkan data ke layar:

1. Perintah "System.out.println(xxx)" akan mencetak data "xxx" ke layar, lalu posisi kursor akan pindah baris. Data berikutnya akan dicetak pada baris di bawahnya.

2. Perintah "System.out.print(xxx)" akan mencetak data "xxx" ke layar, lalu posisi kursor akan berada di samping kanan data terakhir. Data berikutnya akan dicetak di samping data sebelumnya.

Jika data yang akan dicetak lebih dari satu, maka setiap data dipisahkan dengan karakter "+", dengan beberapa kondisi sebagai berikut:

Penulisan	Hasil
`System.out.println("Informasi");`	`Informasi`
`System.out.println(7);`	`7`
`System.out.println("Angka: " + 7);`	`Angka: 7`
`System.out.println(11 + " angka");`	`11 angka`
`System.out.println(5+7);`	`12`
`System.out.println(5 + 7 + " = " + 5 + " + " + 7);`	`12 = 5 + 7`
`System.out.println(5 + " + " + 7 + " = " + (5 + 7));`	`5 + 7 = 12`
`System.out.println(5 + " + " + 7 + " = " + 5 + 7);`	`5 + 7 = 57`

Mengapa hasilnya begini?

Berhati-hatilah jika anda ingin menulis ekspresi penjumlahan sebagai data pada perintah print() maupun println(). Letakkan ekspresi penjumlahan tersebut di dalam pasangan kurung "(" dan ")", sebagaimana dicontohkan pada baris terakhir. Tanda kurung ini tidak diperlukan jika ekspresi penjumlahan itu merupakan data pertama dalam perintah print() maupun println(), sebagaimana dicontohkan pada baris keenam.

2.3 Karakter *Escape*

Karakter *escape* adalah karakter yang memiliki fungsi khusus jika dicetak. Setiap karakter *escape* ditulis dengan didahului oleh karakter *backslash* ("\"). Pada saat sebuah program di-*compile*, ketika compiler Java menjumpai karakter "\", maka karakter berikutnya akan dianggap sebagai karakter *escape*. Jika ternyata karakter berikutnya ini tidak termasuk ke dalam karakter *escape*, maka sebuah pesan error akan ditampilkan.

Java mengenal 7 (tujuh) karakter *escape*, yaitu:

1. Karakter " \n ", menyatakan tanda pindah baris. Data yang terletak sesudahnya akan dicetak pada baris berikutnya.
2. Karakter " \t ", menyatakan tanda tabulasi. Data yang terletak sesudahnya akan dicetak pada tabulasi berikutnya jika mencukupi. Sebuah tabulasi adalah nilai kolom kelipatan 8; dan ini hanya berlaku di layar DOS.
3. Karakter " \b ", menyatakan tanda *backspace*. Data yang terletak sesudahnya akan dicetak pada lokasi satu kolom ke kiri. Jika pada kolom kiri terdapat suatu karakter, maka karakter itu akan diganti dengan karakter pertama dari data yang akan dicetak.
4. Karakter " \r ", menyatakan tanda kolom pertama. Data yang terletak sesudahnya akan dicetak pada kolom pertama namun masih pada baris yang sama.
5. Karakter " \\ ", menyatakan sebuah karakter *backslash*.
6. Karakter " \" ", menyatakan sebuah karakter *double-quote* atau petik ganda.
7. Karakter " \' ", menyatakan sebuah karakter *single-quote* atau petik tunggal.

Contoh pemakaian karakter *escape*:

Program 2.2

```
1  public class Program22
2  {
3    public static void main(String[] args)
4    {
5      System.out.println("Karakter escape");
6      System.out.println("---------------");
7      System.out.println("Karakter escape \n\\n");
8      System.out.println("Karakter escape \t\\t");
9      System.out.println("Karakter escape \b\\b");
10     System.out.println("Karakter escape \r\\r");
11     System.out.println("Karakter escape \\\\");
12     System.out.println("Karakter escape \"\\\");
13     System.out.println("Karakter escape \'\\'");
14   }
15 }
```


Output program:

```
C:\WINDOWS\system32\cmd.exe
Karakter escape
---------------
Karakter escape
\n
Karakter escape          \t
Karakter escape\b
\rrakter escape
Karakter escape \\
Karakter escape "\
Karakter escape '\'
```

Contoh lain:

Program 2.3

```
1  public class Program23
2  {
3    public static void main(String[] args)
4    {
5      System.out.print("Satu\tDua\tTiga\n");
6      System.out.print("Empat\tLima\tEnam");
7    }
8  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Satu    Dua     Tiga
Empat   Lima    Enam
```

Data "Empat Lima Enam" dicetak pada baris kedua meskipun pada baris pertama kita menggunakan perintah print() dan bukannya

println(). Ini disebabkan karena kita melibatkan karakter "\n" di posisi akhir dari data yang dicetak pada print() pertama.

2.4 Tipe Data dan Variabel

Data yang akan, sedang, dan selesai diproses oleh komputer harus disimpan di dalam memori. Agar data yang ada di dalam memori bisa diingat, dibaca, diproses, bahkan dihapus, maka lokasi tempat penyimpanan data itu harus diberi nama karena kita tidak pernah tahu di lokasi mana komputer menyimpan data kita. Konsep ini dikenal dengan istilah *variabel*.

Sebelum bisa digunakan untuk menyimpan data, sebuah variabel harus melalui tahap *deklarasi*, yaitu tahap penentuan jenis data yang bisa disimpan di dalamnya. Sebuah variabel tidak bisa menyimpan lebih dari satu jenis data. Lebih jauh lagi, sebuah variabel juga tidak bisa menyimpan lebih dari satu data dengan jenis yang sama. Kehadiran data berikutnya –dengan jenis yang sama tentu saja—akan menyebabkan data yang telah ada menjadi hilang digantikan oleh data baru tersebut.

Secara umum ada tiga jenis data yang dikenal oleh komputer:

1. Numerik, yaitu data yang berbentuk bilangan, baik bilangan bulat maupun bilangan pecahan.
2. Karakter, yaitu data yang berbentuk karakter tunggal atau deretan karakter.
3. Logika, yaitu data yang berbentuk status benar atau salah.

Java mengenal dua jenis tipe data:

1. *Tipe data primitif*, yaitu tipe data yang diadopsi dari tipe data klasik. Tipe data ini diadopsi dari berbagai bahasa pendahulu Java, antara lain C++ dan Pascal.

2. *Tipe data objek*, yaitu tipe data berbentuk *class* yang merupakan ciri khas dari pemrograman berorientasi objek. Banyak dari tipe data ini yang disediakan untuk mendukung operasional tipe data primitif.

Beberapa tipe data yang sering digunakan adalah sebagai berikut:

Tipe data	Class	Keterangan
int	Integer	Tipe data bilangan bulat
double	Double	Tipe data bilangan pecahan
char	Character	Tipe data karakter tunggal
boolean	Boolean	Tipe data logika
	String	Tipe data string statis
	StringBuffer	Tipe data string dinamis

Sebuah tipe data primitif akan diproses lebih lanjut oleh class yang bersesuaian dengan tipe tersebut, misalnya tipe int akan diproses oleh class `Integer`.

Sebuah variabel dideklarasikan dengan *syntax* sebagai berikut:

```
tipe_data  nama_var [= nilai_awal];
```

Keterangan:

1. Penulisan di dalam kurung siku "**[**" dan "**]**" menyatakan sifatnya *optional*, yaitu boleh ditulis dan boleh tidak ditulis.
2. "nilai_awal" bisa berupa data langsung, ekspresi, atau fungsi.

Contoh:

- boolean status;

- int bilangan = 100;
- double phi = 22.0 / 7;
- double acak = Math.random();

Untuk mendeklarasikan beberapa variabel dengan tipe yang sama, pisahkan antar nama variabel dengan koma.

Contoh multi-deklarasi:

- int angka1, angka2, angka3;
- double phi=22.0/7, acak=Math.random(), nilai=0.0;
- boolean flag=true, status=false;

Java menetapkan bahwa sebuah variabel harus diberi nilai awal. Jika *compiler* Java mendeteksi ada variabel yang dideklarasikan tanpa nilai awal, sebuah pesan error akan ditampilkan dan proses *compile* akan dihentikan, sebagaimana tampak pada contoh program berikut ini:

Program 2.4

```
1  public class Program24
2  {
3    public static void main(String[] args)
4    {
5      int abc;
6
7      System.out.println("isi var abc : " + abc);
8    }
9  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Program2_4.java:7: variable abc might not have been initialized
    System.out.println("isi var abc : " + abc);
                                          ^
1 error
```


Pesan di atas menyatakan bahwa pada file "Program24.java" baris 7, variabel abc belum memiliki nilai, sehingga isi variabel tersebut tidak bisa dicetak.

Pemberian nilai ke dalam sebuah variabel dapat melalui 3 (tiga) cara:

1. Saat *coding* →diketik langsung di dalam program.
2. Saat *start-up* →diketik sebagai parameter program.
3. Saat *running* → diberikan saat berinteraksi dengan user.

Pada saat coding, nilai variabel diberikan secara langsung melalui *assignment-statement*, umumnya menggunakan tanda sama-dengan "=".

Program 2.5

```
1  public class Program25
2  {
3    public static void main(String[] args)
4    {
5      int abc = 50;
6
7      System.out.println("isi var abc : " + abc);
8    }
9  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
isi var abc : 50
```

Perintah Math.random() digunakan untuk meng-*generate* sebuah bilangan acak antara 0 (*inclusive*) sampai 1 (*exclusive*); dengan kata lain tidak sampai 1. Nilai yang dihasilkan bertipe double.

Program 2.6

```
1  public class Program26
2  {
3    public static void main(String[] args)
4    {
5      double abc = Math.random();
6
7      System.out.println("isi var abc : " + abc);
8    }
9  }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
isi var abc : 0.18087552729289424
```

Perintah Math.random() bisa digunakan untuk meng-*generate* sebuah bilangan acak yang berada pada *range* tertentu, misalnya antara 10 sampai 99. Rumus yang digunakan adalah:

```
min + (random() * (max-min+1))
```

Program 2.7

```
1  public class Program27
2  {
3    public static void main(String[] args)
4    {
5      double abc = 10 + Math.random()*90;
6
7      System.out.println("isi var abc : " + abc);
8    }
9  }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
isi var abc : 72.91548265434068
```


Pada saat start-up, variabel diberi nilai melalui parameter program. Parameter program adalah data yang dikirimkan bersamaan dengan penulisan nama program yang akan dieksekusi. Contoh pemakaian parameter program adalah pada saat kita akan meng-*copy* file dari satu lokasi ke lokasi lain.

```
copy *.bat a:
```

Keterangan:

- "`copy`" adalah perintah yang kita berikan ke komputer.
- "`*.bat`" adalah jenis file yang akan disalin, dalam hal ini semua file yang ber-ekstensi .BAT.
- "`a:`" adalah lokasi tujuan, yaitu drive a:.

Kita bisa membuat program Java untuk menjumlahkan 2 bilangan, di mana kedua bilangan tersebut diinputkan sebagai parameter program. Kita bisa saja menjalankan program tersebut dengan *syntax*:

```
C:\>java Jumlahkan 7 5
```

Program akan menjumlahkan bilangan 7 dan 5 lalu menampilkan hasilnya ke layar. Dalam hal ini kedua bilangan ini tidak akan dimasukkan melalui mekanisme "readLine". Berikut ini kode programnya:

Program 2.8

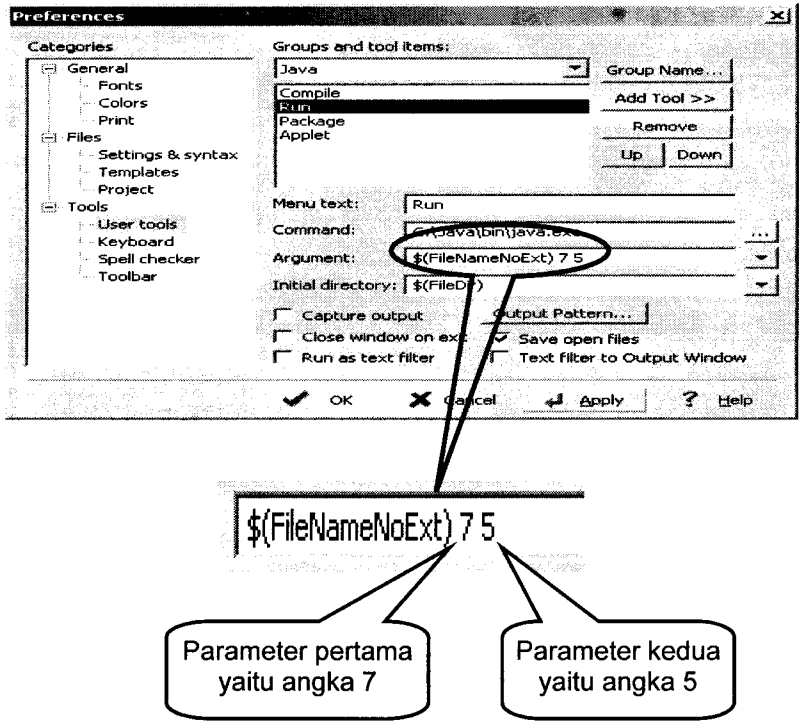
```
1  public class Program28
2  {
3    public static void main(String[] args)
4    {
5      int pertama = Integer.parseInt(args[0]);
6      int kedua = Integer.parseInt(args[1]);
7      int jumlahan = pertama + kedua;
8
9      System.out.println("Bilangan I : " +
   pertama);
10     System.out.println("Bilangan II: " +
   kedua);
```

```
11      System.out.println("Penjumlahan: " +
    jumlahan);
12    }
13  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Bilangan I : 7
Bilangan II: 5
Penjumlahan: 12
```

Jika Anda mengedit kode Java menggunakan EditPlus, maka anda perlu menambah data-data parameter pada setting sebagai berikut:



Parameter tersebut harus diubah secara manual jika Anda bermaksud memasukkan data lain.

Java menangkap parameter yang dikirim ke dalam parameter "args" di dalam fungsi main(). Variabel args ini bertipe *array-of-String*. Parameter pertama memiliki nomor indeks 0 (nol), parameter kedua memiliki nomor indeks 1, dan seterusnya. Anda bisa membaca informasi lebih lanjut mengenai variabel array di bab lain dalam buku ini.
Contoh berikut ini akan menampilkan parameter bertipe String.

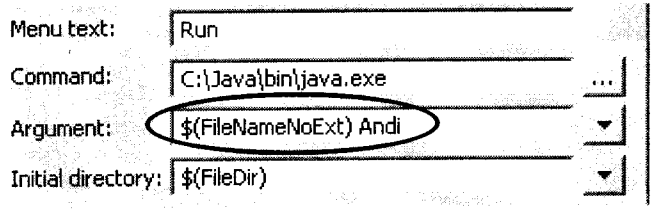
Program 2.9

```
public class Program29
{
  public static void main(String[] args)
  {
    String nama = args[0]);

    System.out.println("Halo " + nama);
  }
}
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Halo Andi
```

Setting di EditPlus adalah sebagai berikut:



Program berikut ini akan menampilkan seluruh parameter yang dikirim, berapa pun jumlahnya.

Program 2.10

```
1  public class Program210
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6      int limit = args.length;
7
8      while (cacah < limit)
9      {
10       System.out.println(args[cacah]);
11       cacah++;
12     }
13   }
14 }
```

Program 2.10 menggunakan instruksi berulang. Topik tentang ini akan dijelaskan pada bab selanjutnya.

Jika program 2.10 dijalankan dengan cara berikut:

```
C:\>java Program210 1 2 3 "belajar java"
```

maka output program adalah:

```
C:\WINDOWS\system32\cmd.exe
1
2
3
belajar java
```

Parameter string yang ditulis di dalam tanda kurung akan dianggap sebagai satu kesatuan data.

Pada saat running, variabel diberi nilai melalui interaksi dengan user. Biasanya data diberikan melalui keyboard, meskipun pada

prakteknya data tersebut bisa diberikan melalui mouse atau alat lain seperti alat pembaca *barcode*.

Class yang disediakan untuk tujuan ini di antaranya adalah DataInputStream dan BufferedReader. Sejak JDK 1.5.0 DataInputStream sudah dianggap sebagai class yang "ketinggalan jaman" atau *deprecated* (tergantikan), sehingga program yang masih menggunakannya akan diberi *note* pada saat compile. Pengguna JDK 1.5.0 atau sesudahnya disarankan untuk menggunakan class BufferedReader.

Program berikut ini menunjukkan cara menerima data *via* keyboard.

Program 2.11

```
1  import java.io.BufferedReader;
2  import java.io.InputStreamReader;
3
4  public class Program211
5  {
6    public static void main(String[] args)
7    {
8      InputStreamReader input = new
   InputStreamReader(System.in);
9      BufferedReader buf = new
   BufferedReader(input);
10
11     try
12     {
13       System.out.print("Siapa nama anda? ");
14       String nama = buf.readLine();
15
16       System.out.print("Tahun berapa anda
   lahir? ");
17       String strTahun = buf.readLine();
18       int tahun = Integer.parseInt(strTahun);
19       int umur = 2007 - tahun;
20
```

```
21          System.out.println("\nHalo " + nama +
", saat ini anda berumur " + umur + " tahun");
22      }
23      catch (Exception e)
24      {
25      }
26    }
27  }
```

Contoh eksekusi program:

```
C:\WINDOWS\system32\cmd.exe
Siapa nama anda? Budi
Tahun berapa anda lahir? 1986

Halo Budi, saat ini anda berumur 21 tahun
```

Method readLine() akan melakukan proses *entry* melalui keyboard. Data yang dimasukkan melalui method ini selalu berupa data string, sehingga jika data yang dimasukkan diharapkan bertipe bilangan, maka harus dilakukan proses konversi data string menjadi data bilangan. Method Integer.parseInt(String) digunakan untuk mengkonversi data string menjadi integer; method Double.parseDouble(String) digunakan untuk mengkonversi data string menjadi double (pecahan presisi ganda). Baris 18 menunjukkan cara mengkonversi data string menjadi integer.

2.5 Blok Program

Sebuah blok program dimulai dengan karakter "{" dan diakhiri dengan karakter "}". Instruksi-instruksi yang ada di dalam sebuah blok program dianggap sebagai satu kesatuan perintah.

Kondisi di mana kehadiran blok program diperlukan:

1. Ketika kita membuat class.
2. Ketika kita membuat method / fungsi.
3. Ketika ada percabangan.
4. Ketika ada perulangan.

Java membolehkan kita membuat blok program pada sembarang kondisi di luar empat kondisi di atas, misalnya jika kita ingin membatasi *scope* variabel atau jika kita ingin mendeklarasikan beberapa variabel independent dengan nama yg sama.

Program 2.12

```
1  public class Program212
2  {
3    public static void main(String[] args)
4    {
5      {
6        int i = 10;
7        System.out.println("\'i\' pada blok 1: " +
i);
8      }
9
10     {
11       int i = 10;
12       System.out.println("\'i\' pada blok 2: " +
i);
13     }
14   }
15 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
'i' pada blok 1: 10
'i' pada blok 2: 40
```

Normalnya, sebuah nama-variabel tidak boleh muncul lebih dari satu kali, akan tetapi dengan adanya blok-program, kita bisa membuat beberapa variabel dengan nama yang sama, seperti ditunjukkan pada program 2.12 di atas.

2.6 Operator

Java menyediakan tiga jenis operator:
1. Operator aritmatika
2. Operator relasi
3. Operator logika

Operator aritmatika adalah operator yang digunakan untuk mengolah data bilangan pada operasi matematika.

Operator	Fungsi	Penggunaan
+	Penjumlahan	5 + 3 menghasilkan 8
-	Pengurangan	5 – 3 menghasilkan 2
*	Perkalian	5 * 3 menghasilkan 15
/	Pembagian	5 / 3 menghasilkan 1 5.0 / 3 menghasilkan 1.66667 5 / 3.0 menghasilkan 1.66667 5.0 / 3.0 menghasilkan 1.66667
%	Modulo (sisa bagi)	5 % 3 menghasilkan 2
&	Operator AND	5 & 3 menghasilkan 1
|	Operator OR	5 | 3 menghasilkan 7
++	Auto-increment	A++ atau ++A menghasilkan isi A ditambah 1
--	Auto-decrement	A-- atau --A menghasilkan isi A dikurangi 1
+=	Penjumlahan	A += B menghasilkan isi A sekarang adalah isi A sebelumnya ditambah dengan isi B
-=	Pengurangan	A -= B menghasilkan isi A sekarang adalah isi A sebelumnya dikurangi dengan isi B

*=	Perkalian	A *= B menghasilkan isi A sekarang adalah isi A sebelumnya dikalikan dengan isi B
/=	Pembagian	A /= B menghasilkan isi A sekarang adalah isi A sebelumnya dibagi dengan isi B
%=	Modulo	A %= B menghasilkan isi A sekarang adalah isi A sebelumnya di-modulo dengan isi B

Hasil dari proses pembagian akan mengikuti tipe data yang terlibat di dalamnya. Jika proses pembagian hanya melibatkan data integer, maka hasilnya akan bertipe integer. Jika salah satu data yang terlibat di dalamnya bertipe pecahan, maka hasilnya akan bertipe pecahan. Java selalu menggunakan tipe double untuk menyatakan data pecahan.

Untuk operator "++" dan "--" berlaku ketentuan sebagai berikut:

- Jika operator tersebut diletakkan di depan nama variabel, maka nilai dari variabel tersebut akan ditambahkan / dikurangi dengan 1 sebelum diproses dengan operasi lain.
- Jika operator tersebut diletakkan di belakang nama variabel, maka nilai dari variabel tersebut akan ditambahkan / dikurangi dengan 1 setelah proses lain selesai dilakukan.

Berikut ini diberikan contoh penggunaan operator auto-increment dan auto-decrement.

```
int a = 10;
int b = ++a;

Hasil akhir:
    • a : 11
    • b : 11
```

```
int a = 10;
int b = a++;

Hasil akhir:
    • a : 11
    • b : 10
```

Isi variabel "a" diberikan ke "b" setelah ditambah 1.	Isi variabel "a" diberikan ke "b" sebelum ditambah 1.

int a = 10;
int b = --a;
 	
Hasil akhir:
• a : 9
• b : 9
 	
Isi variabel "a" diberikan ke "b" setelah dikurangi 1.
	int a = 10;
	int b = a--;
	
	Hasil akhir:
	• a : 9
	• b : 10
	
	Isi variabel "b" diberikan ke "b" sebelum dikurangi 1.

Operator relasi adalah operator yang digunakan untuk mencari hubungan di antara dua data, apakah keduanya sama atau berbeda. Hasil dari operasi relasi adalah salah satu jawaban TRUE atau FALSE, tidak bisa keduanya sekaligus.

Operator	Fungsi	Penggunaan
==	Persamaan	A == B apakah isi variabel A sama dengan isi variabel B?
!=	Pertidaksamaan	A != B apakah isi variabel A tidak sama dengan isi variabel B?
>	Lebih besar dari	A > B apakah isi variabel A lebih besar dari isi variabel B?
>=	Lebih besar dari atau sama dengan	A >= B apakah isi variabel A lebih besar dari atau sama dengan isi variabel B?

<	Lebih kecil dari	A < B apakah isi variabel A lebih kecil dari isi variabel B?
<=	Lebih kecil dari atau sama dengan	A <= B apakah isi variabel A lebih kecil dari atau sama dengan isi variabel B?

Operator logika adalah operator yang digunakan untuk mencari hubungan di antara dua nilai logika. Nilai logika itu sendiri bisa berasal dari variabel boolean maupun dari pernyataan relasi. Hasil dari operasi relasi adalah salah satu jawaban TRUE atau FALSE, tidak bisa keduanya sekaligus, sama seperti operator relasi.

Operator	Fungsi	Penggunaan
!	Logika NOT	!A komplemen (*kebalikan*) dari isi variabel A.
&&	Logika AND	A && B isi variabel A dikenakan operasi AND terhadap dengan isi variabel B.
||	Logika OR	A || B isi variabel A dikenakan operasi OR terhadap dengan isi variabel B.

Berikut ini diberikan tabel kebenaran untuk logika AND dan OR.

X	Y	X and Y	X or Y
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Keterangan:

- 0 – salah
- 1 – benar

Cukup satu kondisi saja yang bernilai salah, maka operasi AND akan bernilai salah. Cukup satu kondisi saja yang bernilai benar, maka operasi OR akan bernilai benar.

Program 2.13

```
1  public class Program213
2  {
3    public static void main(String[] args)
4    {
5      int a = 5;
6      int b = 3;
7
8      System.out.println("Operator Aritmatika");
9      System.out.println("-------------------");
10     System.out.println("   " + a + " + " + b +
                          " = " + (a + b));
11     System.out.println("   " + a + " - " + b +
                          " = " + (a - b));
12     System.out.println("   " + a + " * " + b +
                          " = " + (a * b));
13     System.out.println("   " + a + " / " + b +
                          " = " + (a / b));
14     System.out.println("   " + a + " % " + b +
                          " = " + (a % b));
15     System.out.println("   " + a + " & " + b +
                          " = " + (a & b));
16     System.out.println("   " + a + " | " + b +
                          " = " + (a | b));
17     System.out.println();
18
19     System.out.println("Operator Relasi");
20     System.out.println("---------------");
21     System.out.println("   " + a + " == " + b +
                          " = " + (a == b));
22     System.out.println("   " + a + " != " + b +
                          " = " + (a != b));
23     System.out.println("   " + a + " > " + b +
                          "  = " + (a > b));
```



```
24     System.out.println("   " + a + " >= " + b +
                          " = " + (a >= b));
25     System.out.println("   " + a + " < " + b +
                          "  = " + (a < b));
26     System.out.println("   " + a + " <= " + b +
                          " = " + (a <= b));
27     System.out.println();
28   }
29 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Operator Aritmatika
-------------------
   5 + 3 = 8
   5 - 3 = 2
   5 * 3 = 15
   5 / 3 = 1
   5 % 3 = 2
   5 & 3 = 1
   5 | 3 = 7

Operator Relasi
---------------
   5 == 3 = false
   5 != 3 = true
   5 > 3  = true
   5 >= 3 = true
   5 < 3  = false
   5 <= 3 = false
```

2.7 Konversi Data

Seringkali diperlukan untuk memroses data bertipe tertentu sebagai data bertipe lain, misalnya data string akan diproses sebagai data numerik. Dalam hal ini diperlukan proses konversi data.

Proses konversi data bisa dilakukan dengan dua cara: konvensional dan *type-casting*.

Konversi konvensional

Proses konversi data dilakukan melalui class yang disediakan untuk tipe data yang akan diproses. Class ini menyediakan method khusus yang bertugas mengkonversi data dari tipe lain menjadi data dengan tipe yang ditanganinya. Sebagai contoh, method Integer.parseInt(String) digunakan untuk mengkonversi data string menjadi data integer.

Program 2.14

```
1  public class Program214
2  {
3    public static void main(String[] args)
4    {
5      String str1 = "12345";
6      String str2 = "3.14";
7
8      int bulat = Integer.parseInt(str1);
9      double pecahan = Double.parseDouble(str2);
10
11     System.out.println("   str1 = " + str1);
12     System.out.println("   str2 = " + str2);
13     System.out.println();
14     System.out.println("  bulat = " + bulat);
15     System.out.println("pecahan = " + pecahan);
16     System.out.println();
17   }
18 }
```


Output program:

```
C:\WINDOWS\system32\cmd.exe
   str1 = 12345
   str2 = 3.14

  bulat = 12345
pecahan = 3.14
```

Type Casting

Istilah *type-casting* kurang lebih berarti 'pemaksaan'. Konversi data dengan cara ini akan menyebabkan suatu data –baik data langsung maupun isi variabel—akan mengalami perubahan tipe ketika akan diproses. Jika yang mengalami *type-casting* adalah variabel, maka data aslinya tetap tersimpan dengan tipe asal meskipun ketika akan diproses data tersebut berubah tipe.

Program 2.15

```
1  public class Program215
2  {
3    public static void main(String[] args)
4    {
5      int a = 5;
6      int b = 3;
7
8      System.out.println(a + " / " + b + " = " +
                          a/b);
9      System.out.println();
10   }
11 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
5 / 3 = 1
```

Hasil yang ditampilkan salah karena sifat proses matematika adalah jika data yang terlibat di dalamnya semuanya bertipe bilangan bulat, maka hasilnya akan bertipe bilangan bulat. Berikut ini diberikan perbaikan dari program tersebut.

Program 2.16

```
1  public class Program216
2  {
3    public static void main(String[] args)
4    {
5      int a = 5;
6      int b = 3;
7
8      System.out.println(a + " / " + b + " = " +
                          (double)a/b);
9      System.out.println();
10   }
11 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
5 / 3 = 1.6666666666666667
```

Perbedaan program 2.15 dan program 2.16 bisa dijelaskan sebagai berikut:

- Pada program 2.15, data yang terlibat dalam pembagian semuanya bertipe integer; dengan demikian hasilnya bertipe integer.
- Pada program 2.16, data yang terlibat memang bertipe integer, akan tetapi menjelang proses pembagian dilakukan, isi variabel a diubah sesaat menjadi double, sehingga hasil akhirnya bertipe pecahan.

Metode type-casting banyak digunakan terutama untuk mengakses objek yang bentuknya beragam jika dikaitkan dengan konsep *inheritance*. Topik mengenai *inheritance* bisa Anda baca pada buku kedua.

Berikut ini diberikan contoh program yang melibatkan sejumlah operator yang telah kita pelajari di atas.

Program 2.17

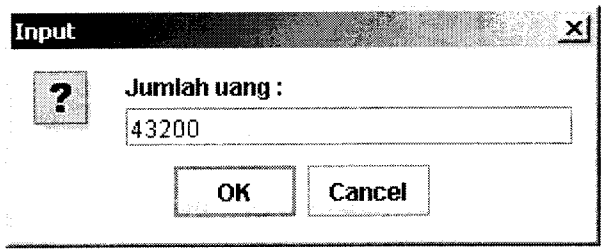
```
1  import javax.swing.JOptionPane;
2
3  public class Program217
4  {
5    public static void main(String[] args)
6    {
7      int seratusRibu = 0;
8      int limapuluhRibu = 0;
9      int duapuluhRibu = 0;
10     int sepuluhRibu = 0;
11     int limaRibu = 0;
12     int seribu = 0;
13     int limaratus = 0;
14     int seratus = 0;
15     int limapuluh = 0;
16     int nilaiUang = 0;
17     int sisa = 0;
18     String input = "";
19     String hasil = "";
20
21     input = JOptionPane.showInputDialog("Jumlah
uang :");
22     nilaiUang = Integer.parseInt(input);
23
24     hasil += "Input = " +
Integer.toString(nilaiUang) + "\n";
```

```
25
26      seratusRibu = nilaiUang / 100000;
27      nilaiUang %= 100000;
28      hasil += "\nSeratus ribu = " +
    Integer.toString(seratusRibu);
29
30      limapuluhRibu = nilaiUang / 50000;
31      nilaiUang %= 50000;
32      hasil += "\nLima puluh ribu = " +
    Integer.toString(limapuluhRibu);
33
34      duapuluhRibu = nilaiUang / 20000;
35      nilaiUang %= 20000;
36      hasil += "\nDua puluh ribu = " +
    Integer.toString(duapuluhRibu);
37
38      sepuluhRibu = nilaiUang / 10000;
39      nilaiUang %= 10000;
40      hasil += "\nSepuluh ribu = " +
    Integer.toString(sepuluhRibu);
41
42      limaRibu = nilaiUang / 5000;
43      nilaiUang %= 5000;
44      hasil += "\nLima ribu = " +
    Integer.toString(limaRibu);
45
46      seribu = nilaiUang / 1000;
47      nilaiUang %= 1000;
48      hasil += "\nSeribu = " +
    Integer.toString(seribu);
49
50      limaratus = nilaiUang / 500;
51      nilaiUang %= 500;
52      hasil += "\nLima ratus = " +
    Integer.toString(limaratus);
53
54      seratus = nilaiUang / 100;
55      nilaiUang %= 100;
56      hasil += "\nSeratus = " +
```

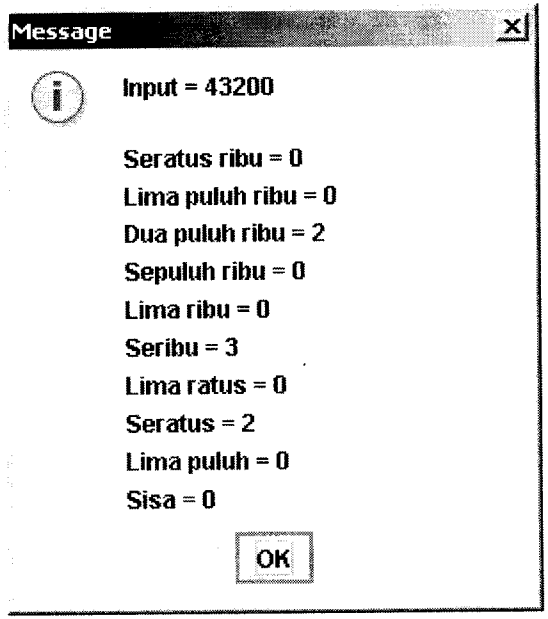


```
    Integer.toString(seratus);
57
58      limapuluh = nilaiUang / 50;
59      nilaiUang %= 50;
60      hasil += "\nLima puluh = " +
    Integer.toString(limapuluh);
61
62      sisa = nilaiUang;
63      hasil += "\nSisa = " +
    Integer.toString(sisa);
64
65      JOptionPane.showMessageDialog(null, hasil);
66      System.exit(0);
67    }
68  }
```

Mula-mula komputer akan bertanya tentang jumlah uang yang akan dihitung:



Kemudian nilai uang tersebut dikonversi menjadi pecahan-pecahan kecil:



2.8 Memformat Tampilan Angka

Penampilan data bilangan secara *tabuler* mengharuskan data bilangan ditampilkan dengan format satuan berjejer dengan satuan, puluhan berjejer dengan puluhan, ratusan berjejer dengan ratusan, dan seterusnya. Secara *default* Java tidak menampilkan bilangan dengan format ini, melainkan ditampilkan dengan sistem "rata-kiri".

Program 2.18

```
1  public class Program218
2  {
3    public static void main(String[] args)
```



```
4    {
5      System.out.println(100);
6      System.out.println(10000);
7      System.out.println(10);
8      System.out.println(1);
9      System.out.println(1000);
10   }
11 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
100
10000
10
1
1000
```

Tampilan seperti ini kurang baik untuk bilangan, apalagi jika bilangan-bilangan tersebut akan diproses, misalnya dijumlahkan. Seharusnya bilangan-bilangan tersebut ditampilkan rata kanan seperti berikut:

```
  100
10000
   10
    1
 1000
```

Java menyediakan class NumberFormat yang bisa digunakan untuk mengatur format penampilan angka. Program 2.18 dimodifikasi sebagai berikut agar menghasilkan output yang baik.

Program 2.19

```
1  import java.text.NumberFormat;
2
3  public class Program219
4  {
```

```
5   public static void main(String[] args)
6   {
7     NumberFormat nf = NumberFormat.getInstance();
8
9     nf.setMinimumIntegerDigits(5);
10
11    System.out.println(nf.format(100));
12    System.out.println(nf.format(10000));
13    System.out.println(nf.format(10));
14    System.out.println(nf.format(1));
15    System.out.println(nf.format(1000));
16  }
17 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
00.100
10.000
00.010
00.001
01.000
```

Tampilan ini menyatakan bahwa:

- Bilangan yang panjangnya kurang dari 5 digit akan mendapat tambahan angka nol di depannya.
- Karakter titik (".") merupakan pemisah ribuan. Ini sesuai dengan *setting* pada komputer kami. Anda mungkin akan mendapat hasil berbeda, misalnya pemisah ribuannya adalah koma (",").

Method setMinimumIntegerDigits(5) digunakan untuk membatas jumlah digit bilangan bulat adalah minimal 5. Ini kita lakukan karena banyak digit maksimum yang akan kita olah adalah 5 yaitu pada angka 10000.

Kita mungkin ingin menghilangkan tanda pemisah ribuan. Method setGroupingUsed(boolean) digunakan untuk mengatur aktivasi fitur tersebut. Jika parameternya *true* maka fitur pemisah ribuan

diaktifkan; sebaliknya jika parameternya *false* maka fitur pemisah ribuan tidak digunakan. Nilai *default* untuk fitur ini adalah *true*.

Program 2.20

```
1  import java.text.NumberFormat;
2
3  public class Program220
4  {
5    public static void main(String[] args)
6    {
7      NumberFormat nf = NumberFormat.getInstance();
8
9      nf.setMinimumIntegerDigits(5);
10     nf.setGroupingUsed(false);
11
12     System.out.println(nf.format(100));
13     System.out.println(nf.format(10000));
14     System.out.println(nf.format(10));
15     System.out.println(nf.format(1));
16     System.out.println(nf.format(1000));
17   }
18 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
00100
10000
00010
00001
01000
```

Program berikut ini akan menunjukkan penggunaan class NumberFormat pada bilangan bertipe pecahan. Pada data pecahan terdapat istilah "integer" dan "fraction". *Integer* adalah data di depan koma dan *fraction* adalah data di belakang koma.

Program 2.21

```
1  import java.text.NumberFormat;
2
3  public class Program221
4  {
5    public static void main(String[] args)
6    {
7      NumberFormat nf = NumberFormat.getInstance();
8
9      nf.setMinimumIntegerDigits(3);
10     nf.setMinimumFractionDigits(5);
11
12     System.out.println(nf.format(100.1));
13     System.out.println(nf.format(3.14));
14     System.out.println(nf.format(22.0/7));
15   }
16 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
100,10000
003,14000
003,14286
```

Method setMinimumFractionDigits(5) digunakan untuk membatas jumlah digit pecahan adalah minimal 5.

Jika angka nol di belakang koma dianggap tidak perlu ada, Anda bisa menggunakan method setMaximumFractionDigits(n) untuk mengatur jumlah digit pecahan yang boleh tampil.

Program 2.22

```
1  import java.text.NumberFormat;
2
3  public class Program222
4  {
5    pul  c static void main(String[] args)
6    {
```



```
7      NumberFormat nf = NumberFormat.getInstance();
8
9      nf.setMaximumFractionDigits(5);
10
11     System.out.println(nf.format(100.1));
12     System.out.println(nf.format(3.14));
13     System.out.println(nf.format(22.0/7));
14   }
15 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
100,1
3,14
3,14286
```

2.9 Class JOptionPane

Sampai saat ini kita sudah membahas tentang cara untuk menerima dan menginformasikan data. Method readLine() digunakan untuk menerima masukan data melalui keyboard, dan method print() atau println() digunakan untuk menampilkan data ke layar. Kedua perintah I/O ini akan bekerja pada layar mode teks, seperti pada saat kita bekerja pada sistem operasi DOS.

Cara lain untuk melakukan proses I/O adalah dengan menggunakan class JOptionPane. Class JOptionPane menyediakan sarana I/O dengan bentuk *interface* berbasis grafik. Model tampilan ini akan mengikuti *interface* yang disediakan oleh sistem operasi.

Class *JOptionPane* terdapat di dalam *package* "javax.swing". Setiap program yang akan menggunakan class ini harus menulis dengan salah satu cara berikut ini:

- Kode `javax.swing.JOptionPane.xxx`, di mana `xxx` adalah fitur dari JOptionPane yang akan digunakan, harus selalu ditulis pada setiap lokasi di mana kita membutuhkan class *JOptionPane*, atau
- Kita menulis `import javax.swing` sekali saja di awal file, lalu pada setiap lokasi yang membutuhkan class *JOptionPane* kita cukup menulis `JOptionPane.xxx`, di mana `xxx` adalah fitur dari *JOptionPane* yang akan digunakan.

Dari sekian banyak fitur yang dimiliki *JOptionPane*, kita hanya akan menggunakan tiga fitur, yaitu showInputDialog(), showMessageDialog() dan showConfirmDialog().

Method showInputDialog()

Method ini digunakan untuk menerima masukan melalui keyboard. Data yang dikembalikan oleh method ini selalu dalam bentuk string. Untuk memperoleh data masukan yang bertipe numerik, kita perlu melakukan konversi data masukan tersebut.

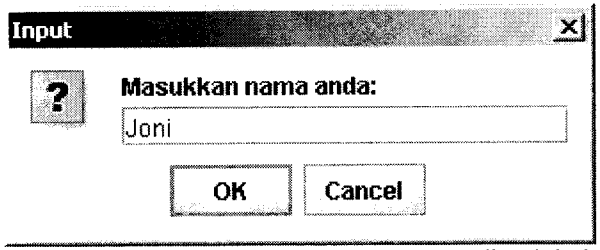
Program 2.23

```
1  import javax.swing.JOptionPane;
2
3  public class Program223
4  {
5    public static void main(String[] args)
6    {
7      String input = JOptionPane.showInputDialog
   ("Masukkan nama anda:");
8      String nama = input;
9
10     input = JOptionPane.showInputDialog
   ("Masukkan tahun lahir anda:");
```

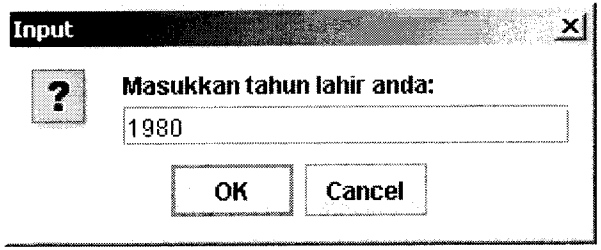


```
11     int lahir = Integer.parseInt(input);
12     int umur = 2007 - lahir;
13
14     System.out.println("Halo " + nama +
   ", umur anda sekarang " + umur + " tahun");
15   }
16 }
```

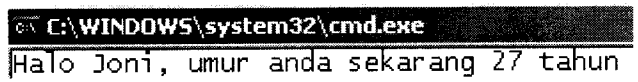
Output program:



Setelah tombol OK di-click, informasi berikut ini akan muncul:



Informasi terakhir yang akan tampil adalah:



Method Integer.parseInt(str) digunakan untuk mengkonversi string str menjadi data bertipe integer. Method Double.parseDouble(str) digunakan untuk mengkonversi string str menjadi data bertipe double / pecahan.

Jika Anda tidak memasukkan data pada form input dialog, hal-hal berikut akan terjadi:

- Data yang dianggap sebagai data masukan adalah *null*.
- Data *null* tidak bisa dikonversi menjadi numerik apa pun.

Jadi pastikan untuk selalu memasukkan data yang diminta, kecuali jika Anda memang bermaksud mengolah data *null* tersebut.

Method showMessageDialog()

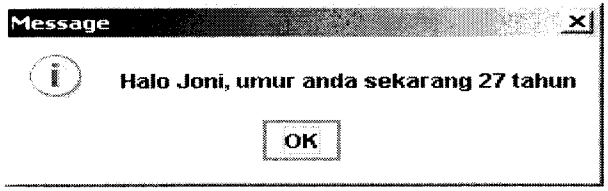
Method ini digunakan untuk menampilkan suatu teks ke layar.

Program 2.24

```
1  import javax.swing.JOptionPane;
2
3  public class Program224
4  {
5    public static void main(String[] args)
6    {
7      String input = JOptionPane.showInputDialog
   ("Masukkan nama anda:");
8      String nama = input;
9
10     input = JOptionPane.showInputDialog
   ("Masukkan tahun lahir anda:");
11     int lahir = Integer.parseInt(input);
12     int umur = 2007 - lahir;
13
14     JOptionPane.showMessageDialog(null, "Halo " +
   nama + ", umur anda sekarang " + umur +
   " tahun");
15   }
16 }
```

Output program:

Dengan data masukan seperti pada program 2.23, form informasi terakhir yang akan tampil adalah:



Parameter pertama method showMessageDialog() adalah nama untuk komponen induk yang memiliki form message tersebut. Kita memberi nilai *null* karena memang tidak ada komponen yang memiliki form tersebut.

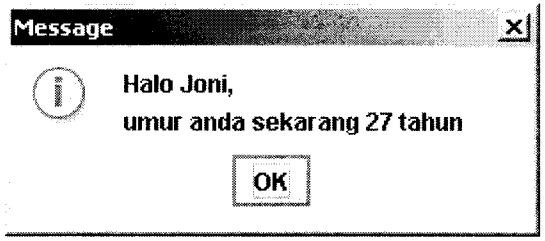
Form message ini bisa menerima karakter "\n" untuk menampilkan teks pada beberapa baris.

Program 2.25

```
1  import javax.swing.JOptionPane;
2
3  public class Program225
4  {
5    public static void main(String[] args)
6    {
7      String input = JOptionPane.showInputDialog
   ("Masukkan nama anda:");
8      String nama = input;
9
10     input = JOptionPane.showInputDialog
   ("Masukkan tahun lahir anda:");
11     int lahir = Integer.parseInt(input);
12     int umur = 2007 - lahir;
13
14     JOptionPane.showMessageDialog(null, "Halo " +
   nama + ",\numur anda sekarang " + umur +
   " tahun");
15   }
16 }
```

Output program:

Dengan data masukan seperti pada program 2.23, form informasi terakhir yang akan tampil adalah:



Kita dapat mengatur judul yang muncul pada form message, dan juga kita dapat mengubah *icon* yang muncul pada form. Pada kondisi *default*, gambar *icon* yang muncul adalah tanda INFORMATION. Java menyediakan beberapa konstanta untuk berbagai maksud penyampaian informasi:

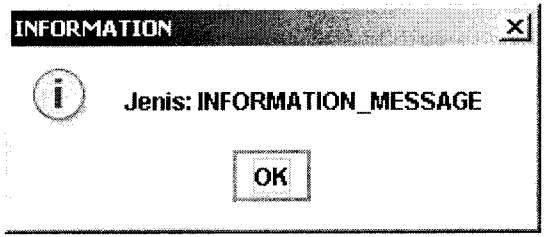
- JOptionPane.INFORMATION_MESSAGE
- JOptionPane.ERROR_MESSAGE
- JOptionPane.WARNING_MESSAGE
- JOptionPane.QUESTION_MESSAGE
- JOptionPane.PLAIN_MESSAGE

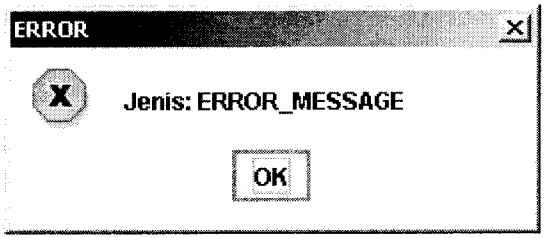
Hendaknya kita menyesuaikan gambar icon dengan jenis message yang akan ditampilkan. Jangan sampai icon ERROR_MESSAGE digunakan untuk menampilkan teks biasa.

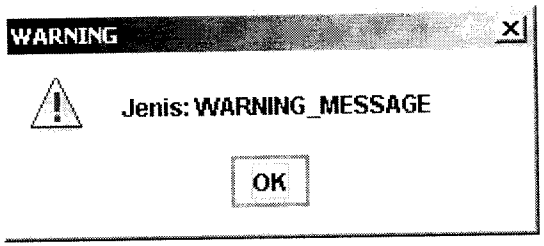
Program 2.26

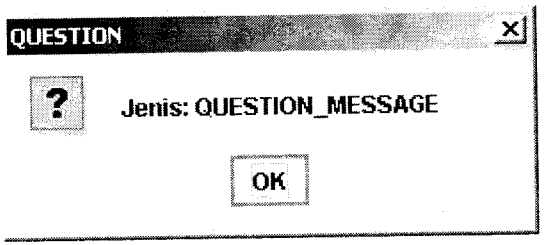
```
1  import javax.swing.JOptionPane;
2
3  public class Program226
4  {
5    public static void main(String[] args)
6    {
7      JOptionPane.showMessageDialog(null,
                  "Jenis: INFORMATION_MESSAGE",
                  "INFORMATION",
                  JOptionPane.INFORMATION_MESSAGE);
8      JOptionPane.showMessageDialog(null,
                  "Jenis: ERROR_MESSAGE",
                  "ERROR",
                  JOptionPane.ERROR_MESSAGE);
9      JOptionPane.showMessageDialog(null,
                  "Jenis: WARNING_MESSAGE",
                  "WARNING",
                  JOptionPane.WARNING_MESSAGE);
10     JOptionPane.showMessageDialog(null,
                  "Jenis: QUESTION_MESSAGE",
                  "QUESTION",
                  JOptionPane.QUESTION_MESSAGE);
11     JOptionPane.showMessageDialog(null,
                  "Jenis: PLAIN_MESSAGE",
                  "PLAIN",
                  JOptionPane.PLAIN_MESSAGE);
12   }
13 }
```

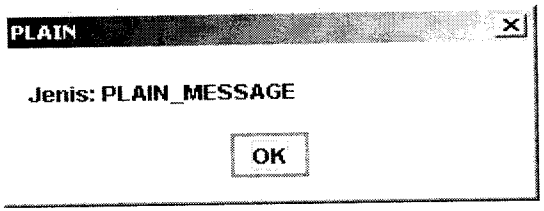
Output program: <akhiri setiap form dengan click tombol OK>











Pada setiap form di atas, coba Anda perhatikan *title* form dan icon yang muncul.

Method showConfirmDialog()

Method ini digunakan untuk melakukan konfirmasi kepada user tentang sesuatu hal, misalnya konfirmasi apakah user akan mengulang suatu proses atau malah akan menutup aplikasi.

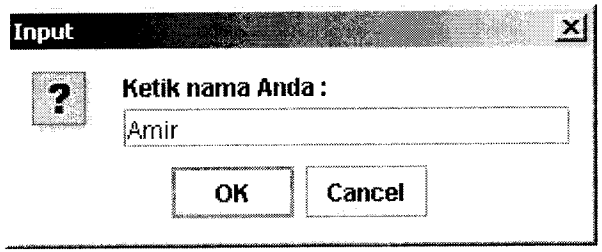
Program 2.27

```
1  import javax.swing.JOptionPane;
2
3  public class Program227
```

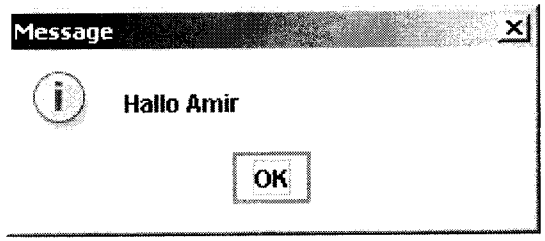


```
4  {
5    public static void main(String[] args)
6    {
7      String nama;
8
9      do
10     {
11       nama = JOptionPane.showInputDialog(
   "Ketik nama Anda :");
12
13       JOptionPane.showMessageDialog(null,
   "Hallo " + nama);
14     }
15     while (JOptionPane.showConfirmDialog(null,
   "Ulangi lagi ?") == JOptionPane.YES_OPTION);
16   }
17 }
```

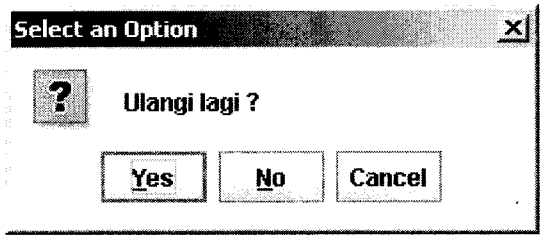
Output program:



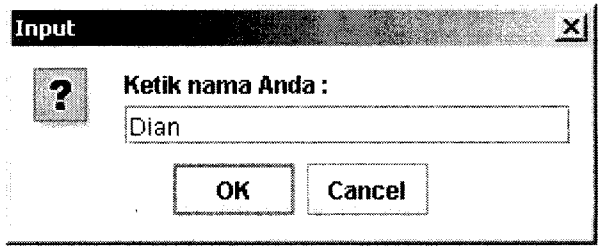
Setelah Anda click OK, tampilan di layar berubah menjadi:



Setelah Anda click OK, tampilan di layar berubah menjadi:



Jika Anda click YES, alur program akan kembali ke tampilan awal yaitu menanyakan nama.



Demikian seterusnya, selama Anda menjawab YES pada form konfirmasi, maka alur program akan balik ke proses awal.

Hasil dari form konfirmasi adalah satu di antara konstanta berikut ini:

- JOptionPane.YES_OPTION → jika yang dipilih adalah YES.
- JOptionPane.NO_OPTION → jika yang dipilih adalah NO.
- JOptionPane.CANCEL_OPTION → jika yang dipilih adalah CANCEL.

2.10 Exception

Exception adalah istilah yang diberikan oleh Java kepada kesalahan yang terjadi di dalam program, yang terjadi pada saat *run-time*. Beberapa kemungkinan error yang bisa terjadi adalah:

- Pemasukan data karakter ketika komputer meminta masukan berupa data bilangan.
- Terjadi pembagian dengan nol.
- Path atau lokasi file yang diberikan tidak sesuai.
- Operasi untuk mengakses variabel array pada nomor indeks di luar batas.

Normalnya, ketika suatu error terjadi pada saat *run-time*, alur proses akan segera terhenti dan komputer menampilkan pesan tertentu yang perlu mendapat perhatian user. Celakanya, data yang sudah dimasukkan ke dalam komputer akan hilang; dan untuk ini user harus memasukkan lagi data-data tersebut.

Java menyediakan mekanisme untuk "menjebak dan menangkap" error yang mungkin terjadi pada saat *run-time*. Setelah error ditangkap, kita bisa menentukan langkah selanjutnya tanpa khawatir kehilangan data yang telah dimasukkan.

Terkait dengan persoalan manajemen exception, Java menyediakan 3 (tiga) metode untuk mengelola exception:

1. Menangkap exception.
2. Membuang exception.
3. Melontarkan exception.

Menangkap exception dilakukan dengan maksud agar alur program tetap dapat dikendalikan meskipun telah terjadi error. Program tidak boleh berhenti mendadak. Aplikasi harus mampu melaporkan error yang sedang terjadi lalu menunggu respon dari user mengenai langkah apa yang harus dilakukan selanjutnya.

Blok `try-catch` digunakan menangkap exception. Strukturnya adalah sebagai berikut:

```
try
{
  ... instruksi yang dikerjakan
      secara normal ...
}
catch (Exception ex)
{
  ... instruksi yang dikerjakan
      jika terjadi error ...
}
```

Seluruh instruksi yang ada di dalam blok *try* akan dikerjakan satu per satu. Selama tidak dijumpai error, instruksi akan berlanjut ke baris berikutnya. Jika suatu ketika terjadi error, kontrol program akan menghentikan eksekusi perintah pada blok *try* lalu beralih ke dalam blok *catch*.

Kita tidak perlu khawatir instruksi blok *try* pada baris berapa error akan terjadi; kita cukup berpikir "jika sampai terjadi error, langkah apa yang harus dikerjakan?" Dengan demikian pikiran kita akan terfokus untuk memikirkan *action* apa yang harus dipersiapkan pada blok *catch*.

Program berikut ini memberikan contoh bagaimana kita bisa menangkap error yang terjadi akibat kesalahan pemasukan data; komputer meminta user memasukkan bilangan, lalu user memasukkan data non-numerik.

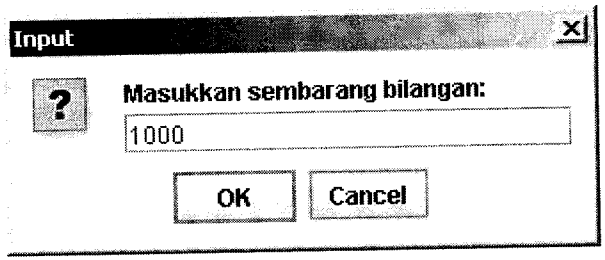
Program 2.28

```
1  import javax.swing.JOptionPane;
2
3  public class Program228
4  {
5    public static void main(String[] args)
6    {
7      try
```

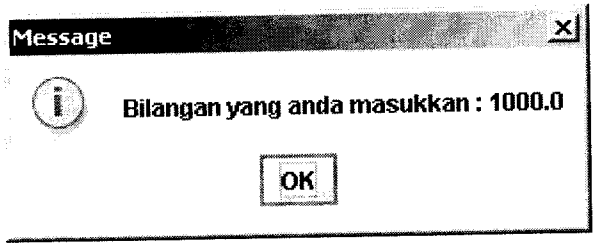


```
8      {
9        String str = JOptionPane.showInputDialog(
   "Masukkan sembarang bilangan");
10       double bil = Double.parseDouble(str);
11
12       JOptionPane.showMessageDialog(null,
   "Bilangan yang anda masukkan : " + bil);
13     }
14     catch (Exception ex)
15     {
16       JOptionPane.showMessageDialog(null,
   "Konversi bilangan gagal.");
17     }
18   }
19 }
```

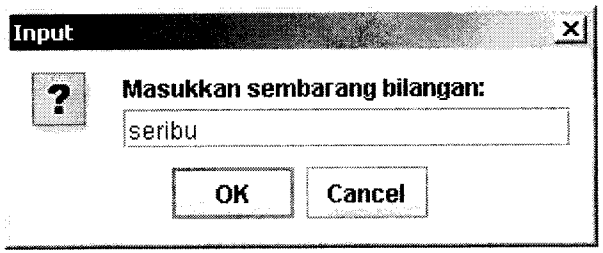
Output program:



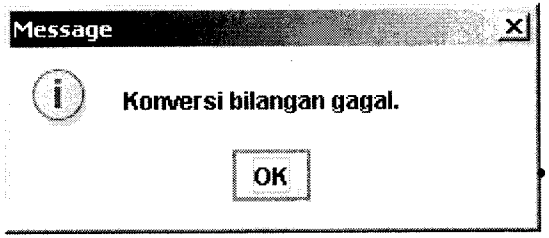
Setelah OK di-click, tampilan di layar berubah menjadi:



Sekarang, jika data yang dimasukkan bukan bertipe numerik:



Setelah OK di-click, tampilan di layar adalah:



Kelebihan dari konsep `try-catch` adalah programmer tidak perlu repot mendeteksi kapan dan di mana suatu error akan terjadi; semua kegiatan itu akan dilakukan oleh komputer. Agak aneh jika kita berpikir untuk mendeteksi setiap baris instruksi untuk memantau apakah terjadi error pada baris itu atau tidak.

Membuang exception dilakukan jika kita ingin mengabaikan exception yang terjadi. Akibat dari terbuangnya exception tersebut, maka program lain yang menjalankan program kita harus melakukan penangkapan exception tersebut. Kegiatan membuang exception ini hanya dilakukan pada level method. Kata kunci `throws` digunakan untuk tujuan ini.

Program 2.29

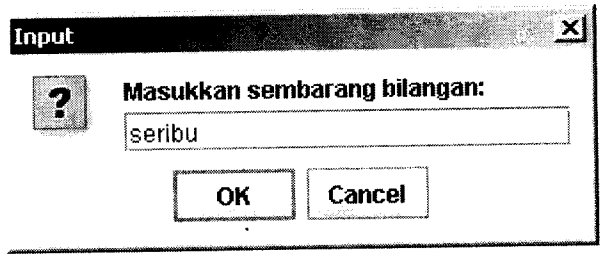
```
1  import javax.swing.JOptionPane;
2
3  public class Program229
4  {
5    public static void main(String[] args)
6    {
```



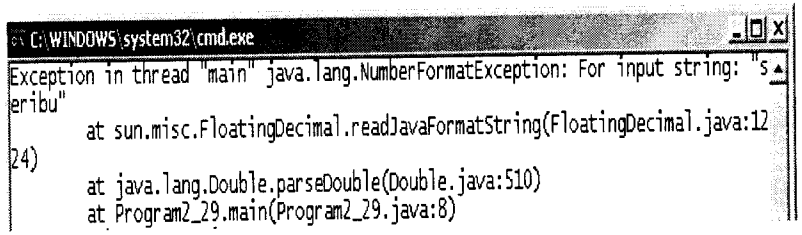
```
7      String str = JOptionPane.showInputDialog(
   "Masukkan sembarang bilangan");
8      double bil = Double.parseDouble(str);
9
10     JOptionPane.showMessageDialog(null,
   "Bilangan yang anda masukkan : " + bil);
11   }
12 }
```

Output program:

Jika data yang dimasukkan bukan bertipe numerik:



Setelah OK di-click, tampilan di layar adalah:



Pesan ini muncul karena tidak ada pihak yang bertugas menangkap error yang dibuang oleh program 2.29, sehingga respon yang diberikan adalah respon alami komputer.

Melontarkan exception dilakukan jika kita ingin memperoleh respon dari luar atas exception yang kita berikan. Melontarkan exception sering dikaitkan dengan pembuatan exception baru akibat dari tidak tersedianya exception yang sesuai dengan kebutuhan kita.

Program berikut ini mendemonstrasikan cara melontarkan exception standar. Exception ini kita lontarkan dalam keadaan program berjalan normal tanpa kesalahan. Kata kunci `throw` digunakan untuk tujuan ini. Kata `throw` tidak diakhiri dengan huruf "s", bedakan dengan `throws` yang bertujuan untuk membuang exception.

Program 2.30

```
1  import javax.swing.JOptionPane;
2
3  public class Program230
4  {
5    public static void main(String[] args)
6    {
7      int a = (int)(Math.random()*100);
8      int a = (int)(Math.random()*100);
9
10     System.out.println("a : " + a);
11     System.out.println("b : " + b);
12     System.out.println();
13
14     if (a > b)
15       throw new Exception();
16   }
17 }
```

Mula-mula nilai variabel a dan b di-*generate* secara acak. Jika ternyata isi variabel a lebih besar daripada isi variabel b, sebuah exception akan dilontarkan.

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
a : 60
b : 27

Exception in thread "main" java.lang.Exception
        at Program2_30.main(Program2_30.java:15)
```


2.11 Konstanta

Variabel yang nilainya tetap sepanjang berjalannya program disebut konstanta. Sebuah konstanta memiliki karakteristik sama seperti variabel biasa, dengan perkecualian bahwa nilai konstanta tidak boleh diubah, apa pun alasannya.

Untuk membedakan antara konstanta dan variabel, Java memberlakukan ketentuan (tidak harus diikuti) sebagai berikut:

1. Nama konstanta ditulis dengan huruf kapital seluruhnya.
2. Jika nama konstanta dua kata atau lebih, maka antar kata dipisah dengan karakter *underscore* (garis bawah, "_").
3. Sebuah konstanta harus langsung diberi nilai pada saat konstanta tersebut didefinisikan.
4. Nilai dari konstanta pada umumnya berlaku global di seluruh class; ini berbeda dengan variabel yang lebih sering digunakan secara lokal.

Kata kunci *final* digunakan untuk mengawali pendefinisian konstanta. Kata ini bisa diterapkan pada beberapa level kode:

1. Level class, maka class ini tidak boleh di-*inherit* menjadi class lain.
2. Level method, maka method (*sub-program*) ini tidak boleh di-*override*.
3. Level variabel, maka variabel ini akan menjadi konstanta. Inilah yang menjadi fokus kita dalam bab ini.

Berikut ini diberikan contoh program yang menggunakan konstanta.

Program 2.31

```
1  import javax.swing.JOptionPane;
2
```

```
3  public class Program231
4  {
5    public static void main(String[] args)
6    {
7      final int MAXIMUM = 10;
8      int cacah = 0;
9
10     while (cacah < MAXIMUM)
11     {
12       System.out.println("Cacah saat ini : " +
   cacah);
13       cacah++;
14     }
15   }
16 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Cacah saat ini : 0
Cacah saat ini : 1
Cacah saat ini : 2
Cacah saat ini : 3
Cacah saat ini : 4
Cacah saat ini : 5
Cacah saat ini : 6
Cacah saat ini : 7
Cacah saat ini : 8
Cacah saat ini : 9
```

Jika Anda mencoba mengubah nilai variabel MAXIMUM maka *compiler* akan melaporkan kesalahan.

```
1  import javax.swing.JOptionPane;
2
3  public class Program231
4  {
5    public static void main(String[] args)
6    {
```



```
7      final int MAXIMUM = 10;
8      int cacah = 0;
9
10     MAXIMUM = 20;
11
12     while (cacah < MAXIMUM)
13     {
14       System.out.println("Cacah saat ini : " +
   cacah);
15       cacah++;
16     }
17   }
18 }
```

Pada saat di-*compile*, Java melaporkan error sebagai berikut:

```
C:\WINDOWS\system32\cmd.exe
Program2_31.java:10: cannot assign a value to final variable MAXIMUM
    MAXIMUM = 20;
    ^
1 error
```

Perubahan nilai konstanta tidak bisa dilakukan di lokasi mana pun, kecuali kode programnya kita ubah.

2.12 Komentar dalam Program

Komentar adalah naskah program yang tidak akan diproses oleh *compiler*. Pada saat proses kompilasi berlangsung, teks program yang termasuk ke dalam komentar akan diabaikan oleh compiler. Sebuah komentar tetap menjadi bagian dari naskah dalam file `.java` tetapi tidak merupakan bagian dari file `.class`.

Kehadiran komentar di dalam program sangat dibutuhkan, terutama jika program yang dibuat sudah masuk ke skala besar dan kompleks. Setidaknya ada 3 (tiga) alasan mengapa komentar perlu ditulis:

1. Dokumentasi
   Pada saat kita sudah mulai membuat program yang besar, adalah perlu untuk memberi keterangan pada beberapa lokasi kode program mengenai kegunaan kode tersebut. Selain itu, dokumentasi juga digunakan untuk menandai tujuan program, siapa pembuatnya, kapan dibuat, dalam rangka apa dibuat, dan sebagainya. Pada saatnya nanti dibutuhkan, maka kita tinggal mencari program dengan kriteria yang telah ditulis sebelumnya.

2. Debugging
   Jika suatu ketika program yang kita buat mengalami kesalahan, maka cara termudah adalah dengan membuang kode program yang kita curigai menjadi lokasi kesalahan tersebut lalu program di-*compile* ulang. Lalu bagaimana jika kemudian diketahui bahwa kode yang telah dihapus itu bukanlah biang kesalahan, namun kesalahan itu ada di tempat lain, apakah Anda akan menulis ulang kode yang telah dihapus itu? Sebenarnya kode tersebut tidak perlu dihapus, tetapi cukup "ditutupi" dengan tanda komentar. Dengan demikian jika di kemudian waktu kode tersebut diperlukan lagi, maka Anda cukup membuka kembali tanda komentar tadi lalu program di-*compile* ulang. Setelah Anda yakin bahwa kode itu adalah sumber masalah, barulah kode itu positif dihapus.

3. Maintenance
   Aplikasi yang dirancang untuk digunakan dalam waktu lama perlu dirawat dan dijaga kesinambungannya. Untuk itu aplikasi yang dibuat perlu diberi informasi yang diperlukan untuk pengembangan periode selanjutnya.

Java menyediakan dua cara menulis komentar:

1. Karakter "//" digunakan untuk mengawali penulisan komentar dalam satu baris. Karakter yang ditulis setelah "//" sampai akhir baris akan diperlakukan sebagai komentar. Cara ini hanya bisa diterapkan pada komentar satu baris. Jika cara ini akan diterapkan pada komentar beberapa baris, maka pada setiap baris komentar karakter "//" harus ditulis di awal komentar.

2. Karakter "/*" digunakan untuk mengawali penulisan komentar dalam satu baris atau lebih, sampai dijumpai karakter "*/". Cara ini memungkinkan kita menulis komentar lebih dari satu baris tanpa harus menulis tanda komentar berulang-ulang. Cukup awali komentar dengan menulis "/*" lalu akhiri komentar dengan menulis "*/".

Program berikut ini akan menunjukkan bagaimana penggunaan komentar dalam program Java.

Program 2.32

```
 1  public class Program232
 2  {
 3    public static void main(String[] args)
 4    {
 5      System.out.println("String ini akan
    ditampilkan.");
 6
 7        //System.out.println("String ini tidak akan
    ditampilkan.");
 8
 9      /*System.out.println("String ini juga tidak
    akan ditampilkan.");*/
10    }
11  }
```

Output program:

2.13 Soal Latihan

1. Buatlah program Java untuk melakukan proses penukaran isi dua buah variabel. Di akhir proses, variabel pertama akan menyimpan isi variabel kedua dan variabel kedua akan menyimpan isi variabel pertama.

2. Buatlah program Java untuk menghitung biaya percakapan telpon jika diketahui biaya yang harus dibayar untuk setiap pulsa telpon dan waktu yang ditetapkan untuk setiap pulsa.

3. Buatlah program Java untuk menghitung waktu yang dibutuhkan untuk menempuh jarak dari tempat satu ke tempat lain dengan kecepatan tertentu.

4. Buatlah program Java untuk menghitung luas dan keliling berbagai benda geometri 2-dimensi, misalnya bujur sangkar.

5. Buatlah program Java untuk menghitung luas dan volume berbagai benda geometri 3-dimensi, misalnya kubus.

BAB – 3
STRUKTUR PERCABANGAN

Tujuan

1. Pembaca memahami konsep struktur kontrol percabangan dalam pemrograman.
2. Pembaca mampu membuat program yang berisi alur program secara bercabang, baik percabangan tunggal, majemuk, maupun bertingkat.
3. Pembaca mampu membuat program yang menggunakan pernyataan relasi maupun pernyataan logika dalam struktur kontrol percabangan.
4. Pembaca mampu mengkombinasikan beberapa pernyataan relasi untuk membentuk sebuah pernyataan logika.
5. Pembaca mampu membuat program yang menggunakan operator "**?**".

3.1 Pengantar

Istilah *percabangan* mengacu kepada perubahan alur proses akibat dari munculnya sejumlah alternatif yang disebabkan oleh perubahan kebutuhan user pada saat sistem sedang berjalan. Pada sistem yang besar, bisa dipastikan bahwa alur proses tidak akan berjalan lurus dari awal sampai akhir, akan tetapi akan dijumpai sejumlah kondisi yang menyebabkan alur proses berbelok, mungkin disebabkan oleh data yang diberikan oleh user, mungkin disebabkan oleh hasil pemrosesan data, dan mungkin disebabkan oleh kombinasi keduanya.

Jika kita ingin mencari analogi dengan peralatan sehari-hari, televisi adalah contoh yang baik untuk menggambarkan bagaimana konsep percabangan diperlukan dalam pemrograman. Sebuah televisi didesain untuk menerima siaran dari berbagai stasiun, tidak peduli berapa gelombangnya. Kita cukup memilih *channel* yang diinginkan, lalu sistem televisi akan menampilkan siaran dari stasiun pilihan kita. Agak aneh jika kita membeli sekian banyak televisi untuk setiap stasiun.

Dalam pemrograman, kita ingin agar program yang dibuat bisa menangani berbagai kondisi, misalnya program itu bisa melakukan konversi bilangan dari sistem Desimal ke dalam sistem Biner, sistem Oktal dan sistem Hexadesimal. Kita bisa saja membuat tiga program: satu untuk konversi Desimal ke Biner, satu untuk konversi Desimal ke Oktal dan satu lagi untuk konversi Desimal ke Hexadesimal, akan tetapi cara ini kurang efisien. Adalah lebih baik jika kita membuat satu program saja namun sudah dilengkapi dengan kemampuan untuk mendeteksi keinginan user, apakah ingin melakukan konversi Desimal ke Biner atau ke yang lainnya.

3.2 Instruksi "if"

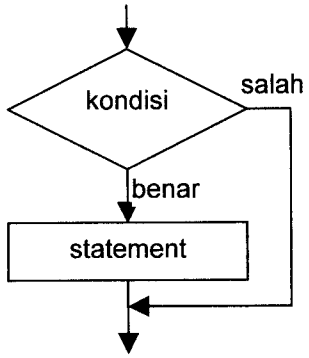
Instruksi yang paling umum digunakan untuk menyatakan percabangan adalah *if*. Bentuk umum dari instruksi *if* adalah:

```
if (kondisi)
{
  statement
}
```

Keterangan:

- **kondisi** adalah pernyataan relasi yang akan diuji kebenarannya. **Kondisi** bisa berbentuk pernyataan relasi tunggal maupun pernyataan relasi majemuk yang dihubungkan dengan operator logika.
- **statement** adalah satu atau lebih perintah yang akan dikerjakan jika **kondisi** bernilai benar.
- Tanda kurung kurawal buka-tutup boleh dihilangkan jika **statement** hanya berupa satu instruksi.

Jika digambarkan dalam bentuk *flowchart*, maka model percabangan *if* bisa digambarkan sebagai berikut:



Gambar 3.1 Flowchart struktur percabangan *if*

Program berikut ini mengilustrasikan pemakaian instruksi *if*.

Program 3.1

```
1  public class Program31
2  {
3    public static void main(String[] args)
4    {
5      int a = (int)(Math.random()*10);
6      int b = (int)(Math.random()*10);
7
8      System.out.println("a = " + a);
9      System.out.println("b = " + b);
10     System.out.print("Kesimpulan: ");
11
12     if (a < b)
13       System.out.println("a < b");
14   }
15 }
```

Program31.java melibatkan fungsi random(), di mana nilai yang akan disimpan oleh variabel a dan b akan di-*generate* secara acak. Setiap kali program dijalankan, nilai pada kedua variabel ini berubah.

Contoh output program #1:

```
C:\WINDOWS\system32\cmd.exe
a = 3
b = 7
Kesimpulan: a < b
```

Munculnya tulisan "Kesimpulan: a < b" ditentukan oleh benar atau salahnya relasi "a < b" pada baris 12. Jika relasi ini bernilai benar maka tulisan "Kesimpulan: a < b" akan ditampilkan; jika salah maka tidak ada yang dikerjakan.

Contoh output program #2:

```
C:\WINDOWS\system32\cmd.exe
a = 9
b = 6
Kesimpulan:
```

Instruksi *if* hanya akan mengerjakan suatu instruksi jika kondisi yang diperiksa bernilai benar; jika kondisi yang diperiksa bernilai salah maka tidak ada yang dikerjakan.

Struktur "if" Multi Kondisi

Kondisi yang diperiksa di dalam perintah *if* bisa berupa kumpulan ekspresi relasi yang dihubungkan dengan satu atau lebih operator logika.

```
if ([!]kondisi1
    [&&/|| [!]kondisi2
    [... ]]])
{
  statement
}
```

Keterangan:

- **kondisi1**, **kondisi2** dan seterusnya adalah beberapa pernyataan relasi yang akan diuji kebenarannya. Hasil akhir dari seluruh kondisi inilah yang akan menentukan apakah blok **statement** akan dikerjakan atau tidak.
- **statement** adalah satu atau lebih perintah yang akan dikerjakan jika **kondisi** bernilai benar.
- Tanda kurung kurawal buka dan tutup boleh dihilangkan jika **statement** hanya berupa satu instruksi.
- Operator "**!**" digunakan untuk membalik suatu kondisi. Jika kondisi ini bernilai salah maka sebelum diproses operator "**!**" akan membaliknya menjadi benar. Jadi yang digunakan dalam proses adalah kondisinya yang bernilai benar.
- Operator "**&&**" digunakan untuk menghubungkan dua relasi dengan logika AND. Hasil akhir akan bernilai benar jika kedua relasi menghasilkan nilai benar; selain itu hasil akhirnya adalah salah.
- Operator "**||**" digunakan untuk menghubungkan dua relasi dengan logika OR. Hasil akhir akan bernilai salah jika kedua relasi menghasilkan nilai salah; selain itu hasil akhirnya adalah benar.

Program berikut ini mengilustrasikan pemakaian instruksi *if* multi kondisi.

Program 3.2

```
1  public class Program32
2  {
3    public static void main(String[] args)
4    {
5      int a = (int)(Math.random()*10);
6      int b = (int)(Math.random()*10);
7      int c = (int)(Math.random()*10);
8
9      System.out.println("a = " + a);
10     System.out.println("b = " + b);
11     System.out.println("c = " + c);
12     System.out.print("Kesimpulan: ");
13
14     if ( (a > b) && (a > c) )
15       System.out.println("nilai 'a' terbesar");
16
17     if ( (b > a) && (b > c) )
18       System.out.println("nilai 'b' terbesar");
19
20     if ( (c > a) && (c > b) )
21       System.out.println("nilai 'c' terbesar");
22   }
23 }
```


Contoh output program #1:

```
C:\WINDOWS\system32\cmd.exe
a = 9
b = 1
c = 2
Kesimpulan: nilai 'a' terbesar
```

Contoh output program #2:

```
C:\WINDOWS\system32\cmd.exe
a = 4
b = 7
c = 5
Kesimpulan: nilai 'b' terbesar
```

Contoh output program #3:

```
C:\WINDOWS\system32\cmd.exe
a = 1
b = 2
c = 4
Kesimpulan: nilai 'c' terbesar
```

Pemakaian fungsi random() merupakan cara yang efektif untuk menguji validitas algoritma dan kode program yang kita buat karena berbagai variasi data bisa muncul. Fungsi ini sangat berguna dalam kondisi sebagai berikut:

- Data yang diperlukan sangat banyak dan hampir mustahil untuk dimasukkan satu per-satu secara manual.
- Data yang digunakan bersifat *simulatif*, artinya data itu diperlakukan sekedar sebagai data tanpa ada interpretasi lain.

Struktur "if" yang Pasti Terjadi

Jika kondisi *if* diberi parameter *true*, maka statement setelahnya pasti akan dikerjakan. Ini sekaligus membuktikan bahwa statement setelah *if* akan dikerjakan jika hasil kondisi yang diperiksa bernilai *true*.

Program 3.3

```
1  public class Program33
2  {
3    public static void main(String[] args)
4    {
5      if (true)
6        System.out.println("Teks ini pasti muncul");
7    }
8  }
```

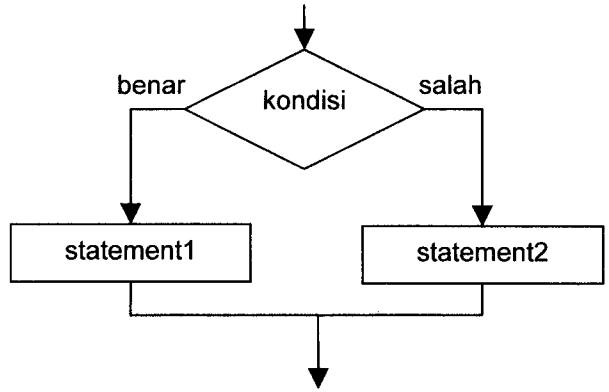
Output program:

```
C:\WINDOWS\system32\cmd.exe
Teks ini pasti muncul
```

Output ini akan selalu muncul tidak peduli berapa kali Anda menjalankan program tersebut.

3.3 Instruksi "if - else"

Konstruksi *if-else* memungkinkan kondisi salah dan benar untuk berisi instruksi, dengan catatan kedua instruksi itu berbeda. Apa pun hasil dari kondisi yang diperiksa, maka selalu ada statement yang dikerjakan.



Gambar 3.2 Flowchart struktur percabangan *if-else*

Jika kondisi bernilai benar maka statement1 akan dikerjakan; dan sebaliknya, jika kondisi bernilai salah maka yang akan dikerjakan adalah statement2.

Program 3.4

```
1  public class Program34
2  {
3    public static void main(String[] args)
4    {
5      int a = (int)(Math.random()*11 - 5);
6
```

```
 7       System.out.println("a = " + a);
 8       System.out.println();
 9
10       if (a < 0)
11         System.out.println("Bilangan " + a +
   " negatif.");
12       else
13       System.out.println("Bilangan " + a +
   " positif.");
14     }
15   }
```

Program34.java akan membangkitkan sebuah bilangan acak antara -5 sampai dengan +5. Rumus yang digunakan tampak pada baris 5, diuraikan sebagai berikut:

```
min + Math.random() * total
```

Rumus ini digunakan untuk membangkitkan bilangan acak mulai dari "min" sebanyak "total" bilangan ke depan. Pada baris 5 kita membangkitkan bilangan acak mulai dari -5 sampai sebanyak 11 bilangan berikutnya, yaitu sampai +5, termasuk di dalamnya ada bilangan nol.

Contoh output program #1:

```
C:\WINDOWS\system32\cmd.exe
a = 3

Bilangan 3 positif.
```

Contoh output program #2:

```
C:\WINDOWS\system32\cmd.exe
a = -1

Bilangan -1 negatif.
```

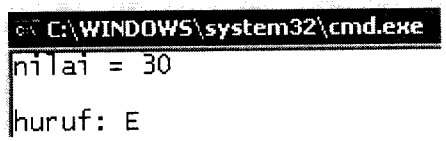

Kalimat "Bilangan positif" dan "Bilangan negatif" tidak akan muncul bersamaan, meskipun keduanya ada pada *if* yang sama. Ini disebabkan karena kita menggunakan instruksi *else* yang bertugas mengerjakan suatu statement jika syarat pada *if* tidak terpenuhi.

Mari kita lihat dua program yang tampak mirip berikut ini, lalu akan kita analisa mana di antara keduanya yang memberikan performa terbaik. Kedua program ini sama-sama bertugas menghitung nilai mahasiswa untuk menjadi sebuah nilai huruf (A, B, C, D atau E).

Program 3.5

```
 1  public class Program35
 2  {
 3    public static void main(String[] args)
 4    {
 5      int nilai = (int)(Math.random()*101);
 6
 7      System.out.println("nilai = " + nilai);
 8      System.out.println();
 9
10      if ( (nilai >= 0) && (nilai < 45) )
11        System.out.println("huruf: E");
12      else
13      if ( (nilai >= 45) && (nilai < 55) )
14        System.out.println("huruf: D");
15      else
16      if ( (nilai >= 55) && (nilai < 65) )
17        System.out.println("huruf: C");
18      else
19      if ( (nilai >= 65) && (nilai < 80) )
20        System.out.println("huruf: B");
21      else
22      if ( (nilai >= 80) && (nilai <= 100) )
23        System.out.println("huruf: A");
24    }
25  }
```

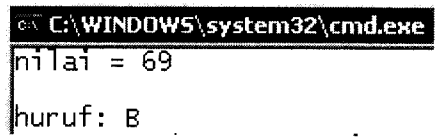
Contoh output program:



Program 3.6

```
1  public class Program36
2  {
3    public static void main(String[] args)
4    {
5      int nilai = (int)(Math.random()*101);
6
7      System.out.println("nilai = " + nilai);
8      System.out.println();
9
10     if ( (nilai >= 0) && (nilai < 45) )
11       System.out.println("huruf: E");
12
13     if ( (nilai >= 45) && (nilai < 55) )
14       System.out.println("huruf: D");
15
16     if ( (nilai >= 55) && (nilai < 65) )
17       System.out.println("huruf: C");
18
19     if ( (nilai >= 65) && (nilai < 80) )
20       System.out.println("huruf: B");
21
22     if ( (nilai >= 80) && (nilai <= 100) )
23       System.out.println("huruf: A");
24   }
25 }
```

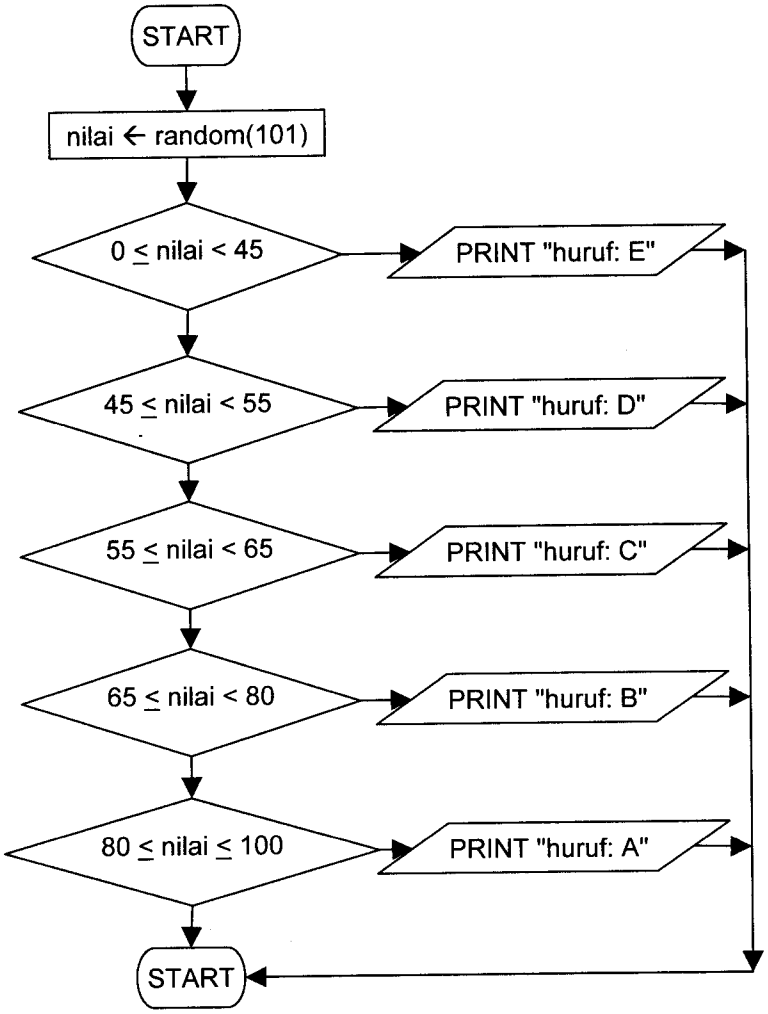

Contoh output program:



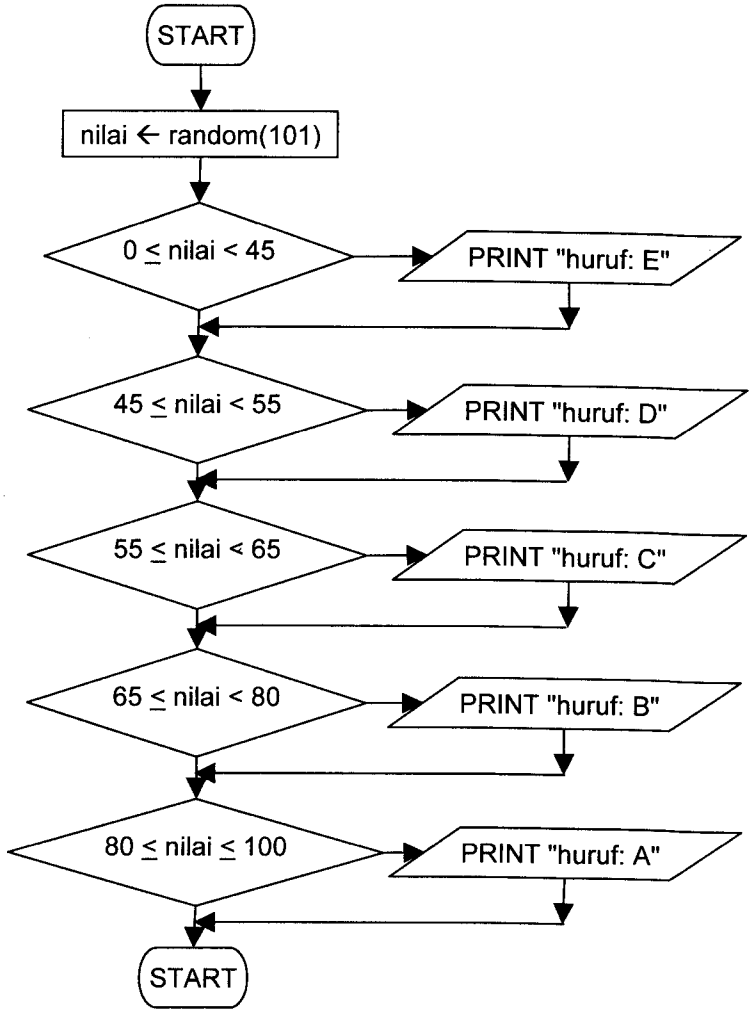
Perbedaan antara Program35.java dan Program36.java adalah salah satunya berisi kata *else* dan lainnya tidak. Hasil keduanya sama saja. Lalu, mana yang lebih baik?

Penggunaan instruksi *else* akan menjamin bahwa setelah sebuah jawaban ditemukan, maka seleksi berikutnya tidak akan dilihat lagi. Alur program akan segera menuju ke instruksi yang terletak setelah blok *if*. Jika kita tidak menggunakan *else* maka meskipun alur program sudah menjumpai jawaban yang benar, alur program tetap akan mengeksekusi blok *if* yang sudah pasti jawabannya salah. Jadi yang terbaik adalah Program35.java.

Jika dibuat bentuk flowchart-nya, maka flowchart kedua program adalah sebagai berikut:



Gambar 3.3 Flowchart untuk Program35.java



Gambar 3.4 Flowchart untuk Program36.java

Program36.java merupakan contoh program yang tidak efektif; meskipun isi variabel nilai sudah berada pada salah satu *range*, komputer tetap memeriksa *range* yang lain.

3.4 Instruksi "switch"

Konstruksi *if-else* yang bertingkat-tingkat seringkali membingungkan pembacaan alur program. Java menyediakan instruksi *switch* untuk memudahkan pembacaan alur program bercabang yang sangat banyak. Meskipun *switch* didesain untuk menggantikan *if-else*, akan tetapi *switch* memiliki batasan:

- Data yang bisa diperiksa haruslah bertipe integer atau char.
- Range data yang bisa diperiksa bernilai 0 s/d 255.

Bentuk umum perintah *switch* adalah sebagai berikut:

```
switch (nilai)
{
  case n1 : statement1;
            break;
  case n2 : statement2;
            break;
  ...
  [default: statementX;]
}
```

Keterangan:

- **nilai** adalah variabel atau ekspresi yang akan diuji.

- *case* **n1**, **n2**, dan seterusnya adalah data yang akan dicocokkan dengan isi **nilai**.
- **statement** adalah pernyataan yang akan dikerjakan jika **nilai** cocok dengan salah satu dari data **n1** atau **n2** atau yang lainnya.
- *break* adalah perintah yang menyatakan akhir dari **statement** yang dikerjakan. Tanpa *break* komputer akan mengerjakan instruksi yang berada di bawahnya walaupun berada di dalam *case* yang lain.
- *default*, bersifat *optional*, dieksekusi jika **nilai** tidak cocok dengan salah satu dari nilai **n1**, **n2**, dan lain-lain yang tersedia. Blok *default* diletakkan di akhir *switch*, dan tidak diakhiri dengan *break*.

Berikut ini diberikan dua buah contoh program dengan hasil sama; program pertama dibuat menggunakan *if-else* dan program kedua menggunakan *switch*. Kedua program ini akan menampilkan nama hari berdasarkan nomor hari yang dibangkitkan secara acak antara 0 s/d 6. Angka 0 berarti hari minggu, 1 berarti senin dan seterusnya.

Program 3.7

```
1  public class Program37
2  {
3    public static void main(String[] args)
4    {
5      int nomor = (int)(Math.random()*7);
6
7      System.out.println("nomor = " + nomor);
8      System.out.print(" hari = ");
9
10     if (nomor == 0)
11       System.out.println("MINGGU");
12     else
13     if (nomor == 1)
14       System.out.println("SENIN");
15     Else
```

```
16      if (nomor == 2)
17        System.out.println("SELASA");
18      else
19      if (nomor == 3)
20        System.out.println("RABU");
21      else
22      if (nomor == 4)
23        System.out.println("KAMIS");
24      else
25      if (nomor == 5)
26        System.out.println("JUMAT");
27      else
28        System.out.println("SABTU");
29    }
30  }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
nomor = 4
hari = KAMIS
```

Program serupa yang menggunakan *switch* adalah sebagai berikut:

Program 3.8

```
1  public class Program38
2  {
3    public static void main(String[] args)
4    {
5      int nomor = (int)(Math.random()*7);
6
7      System.out.println("nomor = " + nomor);
8      System.out.print(" hari = ");
9
10     switch (nomor)
11     {
12       case 0 : System.out.println("MINGGU");
```



```
13               break;
14       case 1 : System.out.println("SENIN");
15               break;
16       case 2 : System.out.println("SELASA");
17               break;
18       case 3 : System.out.println("RABU");
19               break;
20       case 4 : System.out.println("KAMIS");
21               break;
22       case 5 : System.out.println("JUMAT");
23               break;
24       default: System.out.println("SABTU");
25     }
26   }
27 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
nomor = 2
hari = SELASA
```

Apa jadinya jika *break* dihilangkan? Berikut ini contoh programnya:

Program 3.9

```
1  public class Program39
2  {
3    public static void main(String[] args)
4    {
5      int nomor = (int)(Math.random()*7);
6
7      System.out.println("nomor = " + nomor);
8      System.out.print(" hari = ");
9
10     switch (nomor)
11     {
12       case 0 : System.out.println("MINGGU");
13               break;
```

```
14          case 1 : System.out.println("SENIN");
15          case 2 : System.out.println("SELASA");
16          case 3 : System.out.println("RABU");
17                   break;
18          case 4 : System.out.println("KAMIS");
19                   break;
20          case 5 : System.out.println("JUMAT");
21                   break;
22          default: System.out.println("SABTU");
23       }
24    }
25 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
nomor = 1
 hari = SENIN
SELASA
RABU
```

Pada saat variabel nomor bernilai 1, kata yang tercetak seharusnya "SENIN" saja, akan tetapi karena dalam blok ini tidak ada perintah *break*, maka proses dilanjutkan ke baris berikutnya yaitu mencetak kata "SELASA". Pada blok ini pun perintah *break* tidak ada sehingga proses diteruskan ke baris berikutnya yaitu mencetak kata "RABU".

Meskipun kehadiran *break* sangat diperlukan dalam suatu *case*, dalam kondisi tertentu kehadiran *break* memang tidak diharapkan. Berikut ini diberikan program yang memanfaatkan sifat tanpa-*break* tersebut:

Program 3.10

```
1  public class Program310
2  {
3     public static void main(String[] args)
4     {
```



```
5        int nomor = (int)(Math.random()*7);
6
7        System.out.println("nomor = " + nomor);
8        System.out.print("keterangan = ");
9
10       switch (nomor)
11       {
12          case 0 : System.out.println("hari libur");
13                   break;
14          case 1 : ;
15          case 2 : ;
16          case 3 : ;
17          case 4 : ;
18          case 5 : ;
19          case 6 : System.out.println("hari kerja");
20       }
21    }
22 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
nomor = 2
keterangan = hari kerja
```

Pilihan terakhir boleh diakhiri dengan *break* atau tidak ada *break*, hasilnya sama saja.

3.5 Operator "?"

Operator "?" disediakan untuk menggantikan *if-else* yang bertujuan mengisi nilai ke dalam variabel berdasarkan kondisi tertentu. Bentuk umum penggunaan operator "?" adalah:

```
varX = kondisi? nilai1 : nilai2;
```

Variabel varX akan diberi nilai1 jika kondisi bernilai benar, dan akan diberi nilai2 jika kondisi bernilai salah.

Sebagai contoh:

```
if (a > b) z = 0;
   else z = 1;
```

Statement di atas berfungsi mengisi variabel z dengan nilai 0 atau 1 tergantung hasil perbandingan "a>b". Statement ini bisa diganti menggunakan operator "?" dengan syntax sebagai berikut:

```
z = (a > b)? 0 : 1;
```

Operator "?" juga bisa diterapkan pada proses data string, misalnya Program3.10 di atas bisa dibuat sebagai berikut:

Program 3.11

```
1  public class Program311
2  {
3    public static void main(String[] args)
4    {
5      int nomor = (int)(Math.random()*7);
6      String str = (nomor == 0)?
7                   "hari libur" :
8                   "hari kerja";
9
10     System.out.println("nomor = " + nomor);
11     System.out.println("keterangan = " + str);
12   }
13 }
```


Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
nomor = 1
keterangan = hari kerja
```

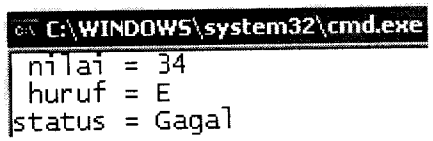
Operator "?" Multi Kondisi

Operator "?" bisa dibuat secara bertingkat, di mana salah satu nilai bisa dibentuk dari operator "?" yang lain. Berikut ini contoh programnya:

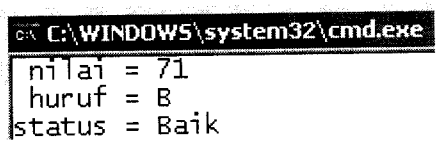
Program 3.12

```
1  public class Program312
2  {
3    public static void main(String[] args)
4    {
5      int nilai = (int)(Math.random()*101);
6
7      char huruf = (nilai < 45)? 'E' :
8                   (nilai < 55)? 'D' :
9                   (nilai < 65)? 'C' :
10                  (nilai < 80)? 'B' : 'A';
11
12     String status = (huruf == 'A')? "Istimewa" :
13                     (huruf == 'B')? "Baik" :
14                     (huruf == 'C')? "Cukup" :
15                     (huruf == 'D')? "Kurang" :
16                                     "Gagal";
17
18     System.out.println(" nilai = " + nilai);
19     System.out.println(" huruf = " + huruf);
20     System.out.println("status = " + status);
21   }
22 }
```

Contoh output program:



Contoh output program:



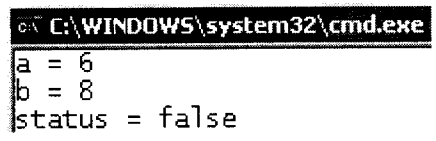
Operator "?" pada Variabel Boolean

Jika operator "?" digunakan untuk memasukkan nilai *true* atau *false* pada sebuah variabel bertipe boolean, maka sebenarnya kita tidak memerlukan operator "?" tersebut.

Program 3.13

```
1  public class Program313
2  {
3    public static void main(String[] args)
4    {
5      int a = (int)(Math.random()*10);
6      int b = (int)(Math.random()*10);
7      boolean status = (a > b)? true : false;
8
9      System.out.println("a = " + a);
10     System.out.println("b = " + b);
11     System.out.println("status = " + status);
12   }
13 }
```


Contoh output program:

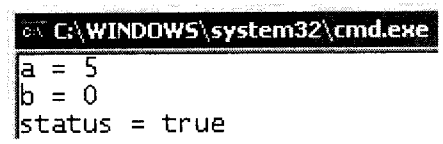


Program di atas bisa dipersingkat menjadi sebagai berikut:

Program 3.14

```
1  public class Program314
2  {
3    public static void main(String[] args)
4    {
5      int a = (int)(Math.random()*10);
6      int b = (int)(Math.random()*10);
7      boolean status = (a > b);
8
9      System.out.println("a = " + a);
10     System.out.println("b = " + b);
11     System.out.println("status = " + status);
12   }
13 }
```

Contoh output program:



Pada baris 7 tampak bahwa isi variabel status akan disesuaikan dengan hasil relasi "a > b".

3.6 Percabangan Secara *Nested*

Sebuah struktur percabangan bisa saja berada di dalam struktur percabangan yang lain, demikian terus tanpa batas, disesuaikan dengan persoalan yang dihadapi. Diproses atau tidaknya percabangan yang berada di bagian dalam ditentukan oleh diproses-tidaknya percabangan bagian luar.

Pada program berikut ini kita akan melihat bagaimana struktur percabangan secara *nested* ini bisa terjadi.

Program 3.15

```
1  public class Program315
2  {
3    public static void main(String[] args)
4    {
5      int angka = (int)(Math.random()*21-10);
6
7      System.out.println("    Angka acak : " +
   angka);
8      System.out.print("Jenis bilangan : ");
9
10     if (angka < 0)
11     {
12       if (angka % 2 == 0)
13         System.out.println("NEGATIF-GENAP");
14       else
15         System.out.println("NEGATIF-GASAL");
16     }
17     else
18     {
19       if (angka % 2 == 0)
20         System.out.println("POSITIF-GENAP");
21       else
22         System.out.println("POSITIF-GASAL");
23     }
```



```
24   }
25 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
    Angka acak : -8
Jenis bilangan : NEGATIF-GENAP
```

Mula-mula komputer akan mengacak sebuah bilangan dalam *range* -10 s/d 10, lalu akan diperiksa apakah angka tersebut kurang dari 0 atau tidak. Apa pun hasil pemeriksaan itu, langkah selanjutnya adalah memeriksa apakah angka tersebut habis dibagi 2 atau tidak. Sebuah bilangan yang habis dibagi 2 pastilah bilangan genap, selain itu termasuk bilangan gasal.

Kadang kala struktur percabangan *nested* seperti itu bisa kita optimasi jika memungkinkan. Berikut ini kita lihat perbaikan dari Program315.

Program 3.16

```
1  public class Program316
2  {
3    public static void main(String[] args)
4    {
5      int angka = (int)(Math.random()*21-10);
6
7      System.out.println("    Angka acak : " +
   angka);
8      System.out.print("Jenis bilangan : ");
9
10     if (angka < 0)
11       System.out.print("NEGATIF-");
12     else
13       System.out.print("POSITIF-");
14
15     if (angka % 2 == 0)
16       System.out.println("GENAP");
```

```
17      else
18        System.out.println("GASAL");
19    }
20  }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
    Angka acak : 6
Jenis bilangan : POSITIF-GENAP
```

3.7 Soal Latihan

1. Buatlah program Java untuk menghitung biaya percakapan telpon berdasarkan lamanya pembicaraan dan jenis sambungan (lokal, SLJJ, SLI).

2. Buatlah program Java untuk melakukan tebak-tebak angka. Komputer akan mengacak sebuah bilangan pada *range* tertentu, lalu user akan mencoba menebak angka tersebut. Jika tebakan benar, komputer akan menampilkan tulisan "ANDA HEBAT", dan jika tebakan salah user diberi kesempatan sampai 3x. Jika selama 3x tebakan salah, komputer akan menampilkan pesan "ANDA KURANG BERUNTUNG".

3. Buatlah program Java untuk melakukan proses konversi nilai suhu pada berbagai satuan: Celcius, Fahrenheit, Reamur, dan Kelvin. Program harus bisa menerima data pada satuan tertentu dan mengkonversinya ke dalam satuan yang lain.

4. Buatlah program Java untuk melakukan proses konversi bilangan pada berbagai sistem basis bilangan: Biner, Oktal, Desimal dan Hexadesimal. Program harus bisa menerima data pada basis bilangan tertentu dan mengkonversinya ke dalam basis bilangan yang lain.

BAB – 4
STRUKTUR PERULANGAN

Tujuan

1. Pembaca memahami konsep struktur kontrol perulangan dalam pemrograman.
2. Pembaca mampu membuat program yang berisi proses berulang menggunakan instruksi *for*.
3. Pembaca mampu membuat program yang berisi proses berulang menggunakan instruksi *while*.
4. Pembaca mampu membuat program yang berisi proses berulang menggunakan instruksi *do-while*.
5. Pembaca mampu menggunakan pernyataan relasi maupun pernyataan logika untuk mengendalikan proses berulang.

4.1 Pengantar

Proses penyelesaian masalah tidak selamanya berjalan lurus. Ada kalanya suatu proses diselesaikan dengan cara yang sama dengan proses sebelumnya namun data yang diolah sedikit berbeda. Dalam hal ini tidak efektif jika kedua cara ini dibuat terpisah.

Jika kita ingin mencetak kata "Java" sebanyak 3 kali, kita bisa saja menulis instruksi sebagai berikut:

```
System.out.println("Java");
System.out.println("Java");
System.out.println("Java");
```

Namun persoalan akan timbul jika kita ingin menampilkan tulisan "Java" lebih dari 3 kali, apakah kita akan menulis instruksi di atas sebanyak yang dibutuhkan, misalnya 100 kali? Karena proses pencetakan "Java" yang pertama, kedua, dan ketiga caranya sama saja, maka seharusnya proses pencetakan cukup dilakukan satu kali lalu cara ini diulangi sebanyak yang dibutuhkan.

Proses berulang dikendalikan oleh variabel yang disebut *counter*, yaitu variabel yang digunakan untuk mencatat jumlah perulangan yang sudah terjadi. Setiap kali sebuah blok instruksi selesai dikerjakan, nilai dalam variabel *counter* diperbarui, biasanya naik 1 nilai. Pada prakteknya nilai variabel *counter* boleh naik atau turun dengan nilai sembarang sesuai kondisi yang dihadapi. Selain berfungsi sebagai penghitung, *counter* tersebut juga bisa digunakan/dilibatkan dalam proses berulang.

Pada perkembangannya, pengendali proses berulang tidak mesti berupa variabel counter. Ada kalanya dibutuhkan pengendali berupa kondisi tertentu. Analoginya, jika seseorang disuruh berlari mengelilingi lapangan sebanyak 5x, maka itu berarti

pengendalinya berupa counter yang akan menghitung jumlah aktivitas mengelilingi lapangan sampai bernilai 5. Akan tetapi jika seseorang disuruh berlari mengelilingi lapangan sampai dia lelah, maka kita tidak tahu sampai berapa kali orang itu akan mengelilingi lapangan; mungkin baru 1x dia sudah lelah, lalu berhenti; atau mungkin sudah 10x dia belum lelah, maka dia terus berlari.

4.2 Instruksi "for"

Java menyediakan perintah *for* yang bisa digunakan untuk mengulang satu atau lebih statement sebanyak jumlah tertentu. Bentuk umum dari instruksi *for* adalah:

```
for ([inisialisasi_counter];
     [syarat_perulangan];
     [update_counter])
{
  statement
}
```

Dari bentuk umum *for* di atas kita bisa lihat bahwa seluruh parameter *for* bersifat *optional*, artinya boleh ditulis dan boleh tidak ditulis.

Andaikan bentuk *for* ditulis sebagai berikut:

```
for (A; B; C)
{
   D;
}
```

maka urutan pengerjaan instruksi *for* adalah sebagai berikut:

- Lakukan A → inisialisasi counter.
- Lakukan B → cek syarat perulangan.
- Lakukan D → jika syarat perulangan terpenuhi.
- Lakukan C → update nilai counter.
- Lakukan B → cek syarat perulangan.
- Lakukan D → jika syarat perulangan terpenuhi.
- Lakukan C → update nilai counter.
- dan seterusnya B, D, C sampai syarat perulangan tidak terpenuhi.

Berikut ini diberikan contoh program yang akan menunjukkan pemakaian perintah *for*.

Program 4.1

```
1  public class Program41
2  {
3    public static void main(String[] args)
4    {
5      int cacah;
6
7      for (cacah=0; cacah<3; cacah++)
8      {
9        System.out.println("Java");
10     }
11   }
12 }
```

Contoh output program:

```
C:\ C:\WINDOWS\system32\cmd.exe
Java
Java
Java
```


Jika statement yang dikerjakan pada blok *for* hanya ada satu, maka tanda kurung kurawal boleh ditiadakan sebagaimana tampak pada program berikut ini.

Program 4.2

```
1  public class Program42
2  {
3    public static void main(String[] args)
4    {
5      int cacah;
6
7      for (cacah=0; cacah<3; cacah++)
8        System.out.println("Java");
9    }
10 }
```

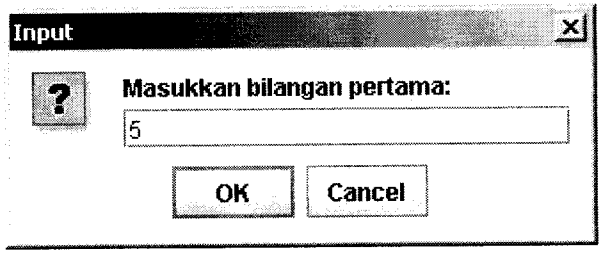
Program berikut ini akan mengimplementasikan perkalian dari sembarang bilangan A dan B. Proses AxB bisa dijabarkan menjadi B+B+ ... +B sebanyak A kali.

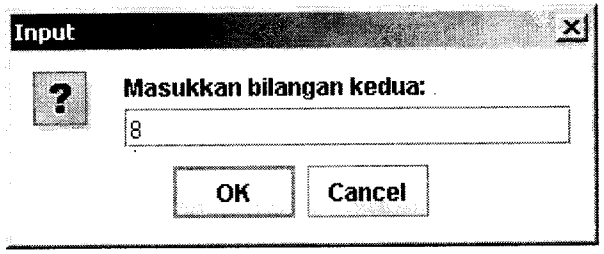
Program 4.3

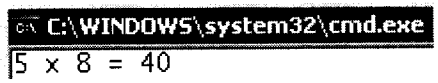
```
1  import javax.swing.JOptionPane;
2
3  public class Program43
4  {
5    public static void main(String[] args)
6    {
7      String s = null;
8
9      s = JOptionPane.showInputDialog("Masukkan
   bilangan pertama:");
10     int a = Integer.parseInt(s);
11
12     s = JOptionPane.showInputDialog("Masukkan
   bilangan kedua:");
13     int b = Integer.parseInt(s);
14
15     int hasil = 0;
```

```
16      for (int i=0; i<a; i++)
17        hasil = hasil + b;
18
19      System.out.println(a + " x " + b + " = " +
    hasil);
20    }
21  }
```

Contoh output program:







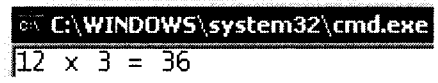
Modifikasi Parameter "for"

Parameter dalam instruksi *for* tidak harus diisi lengkap, bahkan tidak diisi pun tidak apa-apa asalkan karakter titik-koma tetap ditulis. Konsekuensinya, data yang seharusnya terisi sebagai parameter *for* harus ditulis pada baris terpisah di dalam maupun di luar blok *for*.

Program 4.4

```
1   public class Program44
2   {
3     public static void main(String[] args)
4     {
5       int a = (int)(Math.random()*51);
6       int b = (int)(Math.random()*51);
7       int hasil = 0;
8       int i = 0;
9
10      for ( ; i<a; )
11      {
12        hasil = hasil + b;
13        i++;
14      }
15
16      System.out.println(a + " x " + b + " = " +
    hasil);
17    }
18  }
```

Contoh output program:



Baris 10 menunjukkan bagaimana kita bisa mengosongkan beberapa parameter *for*, dengan konsekuensi kita mengisi kekosongan itu di luar dan di dalam blok *for*, yaitu baris 8 dan 13.

Pengaruh Instruksi "break"

Instruksi *break* bisa disisipkan di dalam blok *for* dengan maksud untuk keluar dari blok tersebut. Pemakaian *break* di dalam blok *for* juga terkait erat dengan parameter kedua pada syntax *for*.

Program 4.5

```
1  public class Program45
2  {
3    public static void main(String[] args)
4    {
5      int a = (int)(Math.random()*51);
6      int b = (int)(Math.random()*51);
7      int hasil = 0;
8      int i = 0;
9
10     for ( ; ; )
11     {
12       if (i >= a)
13         break;
14
15       hasil = hasil + b;
16       i++;
17     }
18
19     System.out.println(a + " x " + b + " = " +
   hasil);
20   }
21 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
48 x 24 = 1152
```

Pada baris 10 kita mengosongkan seluruh parameter *for*, khususnya parameter kedua. Baris 12 dan 13 digunakan untuk mengendalikan kapan alur proses akan keluar dari blok *for*.

Contoh lain dari penggunaan *break* bisa dilihat pada program berikut:

Program 4.6

```
1  public class Program46
2  {
3    public static void main(String[] args)
4    {
5      for (int i=0; i<10; i++)
6      {
7        if (i > 5)
8          break;
9
10       System.out.println("i = " + i);
11     }
12   }
13 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
```

Pada saat nilai variabel "i" sudah melebihi 5 alur program keluar dari *for*, sebagaimana tertulis pada baris 7 dan 8.

Pengaruh Instruksi "continue"

Berbeda dengan instruksi *break* yang akan menyebabkan alur proses keluar dari perulangan, instruksi *continue* akan menyebabkan alur kembali ke awal perulangan dengan nilai counter ter-*update*.

Program 4.7

```
1  public class Program47
2  {
3    public static void main(String[] args)
4    {
5      for (int i=0; i<10; i++)
6      {
7        if (i == 5)
8          continue;
9
10       System.out.println("i = " + i);
11     }
12   }
13 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
i = 0
i = 1
i = 2
i = 3
i = 4
i = 6
i = 7
i = 8
i = 9
```

Pada saat nilai variabel "i" sama dengan 5 alur program akan kembali ke awal perulangan lalu memperbarui nilai counter sesuai dengan metode *update* yang ditetapkan, dalam hal ini "i++", lihat baris 5. Dampaknya adalah nilai 5 tidak tercetak ke layar.

Perulangan "for" Multi Kondisi

Blok *for* bisa dikendalikan oleh lebih dari satu kondisi sebagaimana tampak pada contoh-contoh di atas. Parameter kedua pada *for* bisa diisi dengan lebih dari satu kondisi.

Program 4.8

```
1  public class Program48
2  {
3    public static void main(String[] args)
4    {
5      for (int i=0, j=15;
6           (i<10) && (j>10);
7           i++, j--)
8      {
9        System.out.println("i = " + i + ",  j = "
   + j);
10     }
11   }
12 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
i = 0,  j = 15
i = 1,  j = 14
i = 2,  j = 13
i = 3,  j = 12
i = 4,  j = 11
```

Kami sengaja menulis parameter *for* dalam tiga baris: baris 5 untuk parameter pertama, baris 6 untuk parameter kedua dan baris 7 untuk parameter ketiga. Parameter pertama dan ketiga boleh diisi lebih dari satu data yang dipisahkan dengan koma, sedangkan parameter kedua boleh diisi lebih dari satu data dengan pemisah operator logika.

Pada baris 6 tampak bahwa syarat terjadinya perulangan ditentukan oleh dua kondisi yang masing-masing dikendalikan oleh variabel "i" dan "j". Variabel "i" diperbarui secara menaik dan variabel "j" diperbarui secara menurun.

4.3 Instruksi "while"

Instruksi *while* digunakan untuk melakukan proses berulang jika syaratnya terpenuhi. Sebelum blok statement dikerjakan, terlebih dahulu syaratnya diperiksa; jika syaratnya terpenuhi maka blok statement akan dikerjakan.

Bentuk umum dari instruksi *while* adalah:

```
while (kondisi_terpenuhi)
{
  statement
}
```

Dari bentuk umum *while* di atas bisa dilihat bahwa blok statement belum tentu akan dikerjakan. Jika di awal pemeriksaan ternyata kondisi bernilai salah (*false*) atau tidak terpenuhi, maka blok statement tidak akan dikerjakan.

Berikut ini diberikan beberapa contoh program penggunaan *while*.

Program 4.9

```
1  public class Program49
2  {
3    public static void main(String[] args)
4    {
5      int cacah=0;
6
7      while (cacah < 5)
8      {
9        System.out.println("Java");
10       cacah++;
11     }
12   }
13 }
```

Output program:

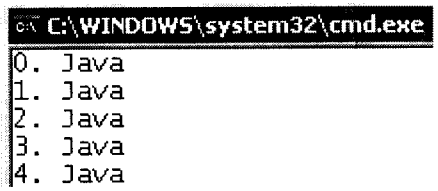
```
C:\WINDOWS\system32\cmd.exe
Java
Java
Java
Java
Java
```

Pada program di atas, selama variabel cacah bernilai di bawah 5, blok statement baris 9 sampai 10 akan dikerjakan.

Program 4.10

```
1  public class Program410
2  {
3    public static void main(String[] args)
4    {
5      int cacah=0;
6
7      while (cacah < 5)
8      {
9        System.out.println(cacah + ". Java");
10       cacah++;
11     }
12   }
13 }
```

Output program:

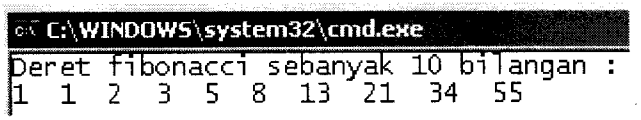


Pada program di atas variabel cacah ikut dilibatkan dalam proses berulang, yaitu ikut dicetak ke layar.

Program 4.11

```
1  public class Program411
2  {
3    public static void main(String[] args)
4    {
5      int kiri = 0;
6      int kanan = 1;
7      int banyak = 10;
8      int hitung = 0;
9
10     System.out.println("Deret fibonacci sebanyak
" + banyak + " bilangan :");
11
12     while (hitung < banyak)
13     {
14       System.out.print(kanan + "  ");
15
16       int bantu = kiri + kanan;
17       kiri = kanan;
18       kanan = bantu;
19
20       hitung++;
21     }
22   }
23 }
```


Output program:

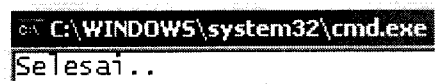


Program di atas akan menampilkan deret Fibonacci sebanyak *N* bilangan pertama, dimana nilai *N* ditentukan melalui variabel banyak.

Program 4.12

```
1  public class Program412
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 5;
6
7      while (cacah < 5)
8      {
9        System.out.println("Java");
10       cacah++;
11     }
12
13     System.out.println("Selesai..");
14   }
15 }
```

Output program:



Pada program di atas tulisan "Java" tidak pernah muncul karena syarat untuk munculnya tulisan itu tidak pernah terpenuhi bahkan sejak awal proses. Nilai variabel cacah yang tidak pernah di bawah 5 menyebabkan tulisan "Java" tidak pernah muncul. Kondisi ini kontradiksi dengan perulangan berikutnya, *do-while*.

Pengaruh Instruksi "break"

Instruksi *break* pada blok perulangan *while* akan mengalihkan alur keluar dari blok ini. Biasanya, meski tidak harus, instruksi *break* digunakan bersamaan dengan *if*.

Program 4.13

```
1  public class Program413
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6
7      while (cacah < 10)
8      {
9        if (cacah > 5)
10         break;
11
12       System.out.println(cacah + ". Java");
13       cacah++;
14     }
15
16     System.out.println("Selesai..");
17   }
18 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
0. Java
1. Java
2. Java
3. Java
4. Java
5. Java
Selesai..
```


Meskipun pada baris 7 tertulis bahwa perulangan akan terjadi 10x, akan tetapi instruksi pada baris 9 dan 10 menyebabkan proses terhenti pada hitungan ke-5.

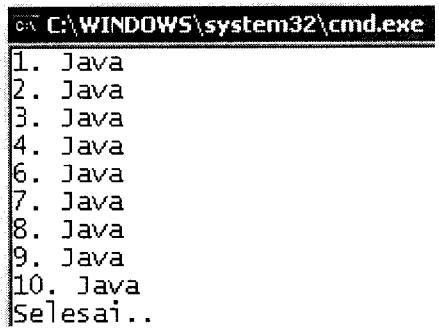
Pengaruh Instruksi "continue"

Instruksi *continue* juga bisa berpengaruh pada perulangan *while*. Hal utama yang perlu diperhatikan adalah lokasi *continue* haruslah berada setelah proses *update* counter, karena jika tidak demikian akan terjadi proses berulang tanpa henti.

Program 4.14

```
1  public class Program414
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6
7      while (cacah < 10)
8      {
9        cacah++;
10
11       if (cacah == 5)
12         continue;
13
14       System.out.println(cacah + ". Java");
15     }
16
17     System.out.println("Selesai..");
18   }
19 }
```

Output program:



Instruksi pada baris 11 dan 12 menyebabkan ketika variabel cacah bernilai 5 alur proses akan kembali ke awal perulangan (baris 7) dengan membawa nilai cacah saat itu yaitu 5. Akibatnya, nilai 5 tidak tercetak di layar. Apa yang akan terjadi jika statement "cacah++" diletakkan setelah *continue*?

Perulangan "while" Multi Kondisi

Perulangan *while* multi kondisi akan menggunakan beberapa kondisi sekaligus untuk menentukan apakah suatu perulangan akan terjadi atau tidak. Beberapa kondisi ini akan dihubungkan dengan operator logika sebagaimana pada instruksi *for*. Untuk memudahkan pembacaan program, sebaiknya setiap kondisi ditulis di dalam pasangan tanda-kurung.

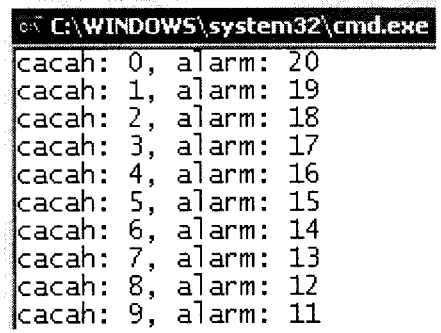
Program 4.15

```
1  public class Program415
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6      int alarm = 20;
7
8      while ( (cacah < 10) && (alarm > 0) )
```



```
9      {
10       System.out.println("cacah: " + cacah +
", alarm: " + alarm);
11       cacah++;
12       alarm--;
13     }
14   }
15 }
```

Output program:



Selama variabel cacah berada di bawah 10 dan variabel alarm berada di atas 0, program akan mencetak suatu string. Jika program tersebut kita modifikasi menggunakan operator logika OR, maka hasilnya adalah sebagai berikut:

Program 4.16

```
1  public class Program416
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6      int alarm = 20;
7
8      while ( (cacah < 10) || (alarm > 0) )
9      {
10       System.out.println("cacah: " + cacah +
```

```
", alarm: " + alarm);
        cacah++;
        alarm--;
      }
    }
}
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
cacah: 0, alarm: 20
cacah: 1, alarm: 19
cacah: 2, alarm: 18
cacah: 3, alarm: 17
cacah: 4, alarm: 16
cacah: 5, alarm: 15
cacah: 6, alarm: 14
cacah: 7, alarm: 13
cacah: 8, alarm: 12
cacah: 9, alarm: 11
cacah: 10, alarm: 10
cacah: 11, alarm: 9
cacah: 12, alarm: 8
cacah: 13, alarm: 7
cacah: 14, alarm: 6
cacah: 15, alarm: 5
cacah: 16, alarm: 4
cacah: 17, alarm: 3
cacah: 18, alarm: 2
cacah: 19, alarm: 1
```

Mengapa hasil Program415 dan Program416 tersebut berbeda?

Auto-update Counter "while"

Perubahan nilai counter pada perulangan *while* bisa terjadi secara *auto*, di mana perubahan itu terjadi secara otomatis dan tersamar di dalam statement *while*. Untuk mereka yang terbiasa menggunakan bahasa pemrograman selain C++ atau Java, cara

ini akan sedikit membingungkan dan menyulitkan saat dilakukan pembacaan dan penelusuran alur program. Auto-update ini bisa berupa *increment* maupun *decrement*, dan lokasi yang digunakan bisa secara *pre* maupun *post*. Anda bisa berlatih dengan berbagai variasi fitur ini.

Program 4.17

```
public class Program417
{
  public static void main(String[] args)
  {
    int cacah = 0;

    while (cacah++ < 10)
      System.out.println("cacah: " + cacah);
  }
}
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
cacah: 1
cacah: 2
cacah: 3
cacah: 4
cacah: 5
cacah: 6
cacah: 7
cacah: 8
cacah: 9
cacah: 10
```

Mengapa nilai 0 tidak tercetak sedangkan nilai 10 tercetak?

Sebagaimana pada *for*, jika instruksi dalam *while* hanya ada satu maka pasangan tanda kurung-kurawal bisa dihilangkan.

Program 4.18

```
1  public class Program418
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6
7      while (++cacah < 10)
8        System.out.println("cacah: " + cacah);
9    }
10 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
cacah: 1
cacah: 2
cacah: 3
cacah: 4
cacah: 5
cacah: 6
cacah: 7
cacah: 8
cacah: 9
```

Mengapa nilai 0 dan 10 sama-sama tidak tercetak?

Pada Program417 dan Program418 kita melakukan perubahan nilai counter cacah langsung di dalam statement *while*, yaitu pada baris 7.

4.4 Instruksi "do-while"

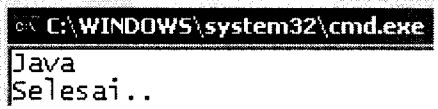
Struktur perulangan *do-while* akan menjamin bahwa instruksi yang akan diulang pasti akan dikerjakan minimal satu kali. Ini berbeda dengan *while* yang tidak menjamin dieksekusinya instruksi berulang. Jika kita analogikan dalam kehidupan sehari-hari, blok *while* adalah ibarat petugas satpam yang berjaga-jaga di depan rumah. Tamu yang datang berkunjung belum tentu diperbolehkan masuk. Jika tamu tersebut dicurigai berbahaya, maka satpam tidak mengijinkannya masuk; sebaliknya, jika semua dalam kondisi baik maka si tamu diperbolehkan masuk.

Blok *do-while* bisa diibaratkan sebagai tempat penitipan sepeda motor. Setiap sepeda motor boleh dititipkan di dalamnya, akan tetapi tidak setiap sepeda motor yang dititipkan diperbolehkan keluar. Jika sepeda motor yang dititipkan tidak dilengkapi dengan surat-surat seperti STNK, maka selama surat-surat itu belum lengkap sepeda motor itu tidak diperbolehkan keluar area penitipan.

Program 4.19

```
1  public class Program419
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 5;
6
7      do
8      {
9        System.out.println("Java");
10       cacah++;
11     }
12     while (cacah < 5);
13
14     System.out.println("Selesai..");
15   }
16 }
```

Output program:



Bandingkan dengan Program412 yang telah kita buat sebelumnya menggunakan blok perulangan *while*. Pada Program412 tidak ada tulisan "Java" di layar, sedangkan pada Program419 di atas kita memperoleh tulisan "Java" tercetak sebanyak 1x.

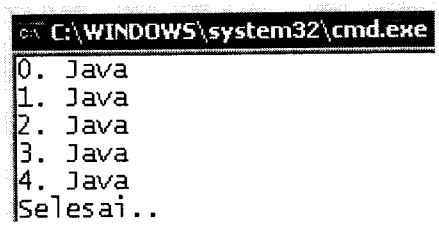
Pengaruh Instruksi "break"

Instruksi *break* juga bisa diterapkan pada perulangan *do-while* untuk tujuan mengalihkan alur program keluar dari blok berulang.

Program 4.20

```
1  public class Program420
2  {
3    public static void main(String[] args)
4    {
5      Int cacah = 0;
6
7      Do
8      {
9        if (cacah == 5)
10         break;
11
12       System.out.println("Java");
13       cacah++;
14     }
15     while (cacah < 10);
16
17     System.out.println("Selesai..");
18   }
19 }
```


Output program:



Proses berulang akan berhenti ketika variabel cacah mencapai nilai 5 meskipun batas yang ditentukan adalah 10.

Pengaruh Instruksi "continue"

Instruksi *continue* bisa diterapkan pada perulangan *do-while* dengan maksud untuk melanjutkan proses berulang dengan nilai counter yang baru. Instruksi *continue* harus diletakkan setelah proses update counter.

Program 4.21

```
1  public class Program421
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6
7      do
8      {
9        cacah++;
10
11       if (cacah == 5)
12         continue;
13
14       System.out.println(cacah + ". Java");
15     }
16     while (cacah < 10);
```

```
17
18      System.out.println("Selesai..");
19    }
20  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
1. Java
2. Java
3. Java
4. Java
6. Java
7. Java
8. Java
9. Java
10. Java
Selesai..
```

Ketika variabel cacah mencapai nilai 5, proses berulang akan kembali ke baris 7 dan mengabaikan baris 14, sehingga tampak bahwa nilai 5 tidak tercetak di layar.

Perulangan "do-while" Multi Kondisi

Perulangan *do-while* yang multi kondisi bisa dicontohkan sebagai berikut:

Program 4.22

```
1  public class Program422
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6      int alarm = 20;
7
8      do
9      {
```



```
10       System.out.println("cacah: " + cacah +
", alarm: " + alarm);
11       cacah++;
12       alarm--;
13     }
14     while ( (cacah < 10) && (alarm > 0) );
15   }
16 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
cacah: 0, alarm: 20
cacah: 1, alarm: 19
cacah: 2, alarm: 18
cacah: 3, alarm: 17
cacah: 4, alarm: 16
cacah: 5, alarm: 15
cacah: 6, alarm: 14
cacah: 7, alarm: 13
cacah: 8, alarm: 12
cacah: 9, alarm: 11
```

Auto-update Counter "do-while"

Proses update counter pada *do-while* juga bisa dilakukan secara otomatis pada statement *do-while*. Berikut ini contoh programnya:

Program 4.23

```
1  public class Program423
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6      int alarm = 20;
7
8      do
9      {
```

```
10          System.out.println("cacah: " + cacah +
", alarm: " + alarm);
11      }
12      while ( (cacah++ < 10) && (--alarm > 0) );
13    }
14  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
cacah: 0, alarm: 20
cacah: 1, alarm: 19
cacah: 2, alarm: 18
cacah: 3, alarm: 17
cacah: 4, alarm: 16
cacah: 5, alarm: 15
cacah: 6, alarm: 14
cacah: 7, alarm: 13
cacah: 8, alarm: 12
cacah: 9, alarm: 11
cacah: 10, alarm: 10
```

Kami persilahkan Anda menelusuri jalannya program tersebut sehingga tampilan di layar adalah seperti itu.

4.5 Perulangan Secara *Nested*

Istilah *nested-loop* adalah situasi di mana sebuah proses berulang terletak di dalam proses berulang lainnya. Nested-loop itu sendiri tidak terikat pada satu jenis instruksi berulang, jadi di dalam instruksi *for* bisa ada *for* yang lain atau *while* atau *do-while*. Demikian juga di dalam *while* atau *do-while* bisa terdapat instruksi *for* atau *while* atau *do-while* lainnya.

Program berikut ini akan menampilkan struktur nomor urut beserta sub-nomornya.

Program 4.24

```
1  public class Program424
2  {
3    public static void main(String[] args)
4    {
5      for (int i=1; i<=3; i++)
6      {
7        System.out.println(i + ".");
8
9        for (int j=1; j<=5; j++)
10         System.out.println("   " + i + "." +
j + ".");
11     }
12   }
13 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
1.
   1.1.
   1.2.
   1.3.
   1.4.
   1.5.
2.
   2.1.
   2.2.
   2.3.
   2.4.
   2.5.
3.
   3.1.
   3.2.
   3.3.
   3.4.
   3.5.
```

Program selanjutnya berguna untuk menampilkan pola bintang berdasarkan jumlah baris yang ditentukan.

Program 4.25

```
1  public class Program425
2  {
3    public static void main(String[] args)
4    {
5      int baris = 5;
6
7      for (int i=0; i<baris; i++)
8      {
9        for (int j=0; j<=i; j++)
10         System.out.print("x ");
11
12       System.out.println();
13     }
14   }
15 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
x
x x
x x x
x x x x
x x x x x
```

Tampilan di atas adalah untuk nilai baris 5.

Program berikut ini akan menampilkan pola bintang dengan model tampilan segitiga puncak di tengah. Jumlah baris di-*generate* secara acak antara 1 s/d 20.

Program 4.26

```
1  public class Program426
2  {
```



```
3    public static void main(String[] args)
4    {
5      int baris = (int)(Math.random()*20+1);
6
7      for (int i=0; i<baris; i++)
8      {
9        for (int j=0; j<baris-i-1; j++)
10         System.out.print(" ");
11
12       for (int j=0; j<=i; j++)
13         System.out.print("x ");
14
15       System.out.println();
16     }
17
18     System.out.println("\nbaris = " + baris);
19   }
20 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
                    x
                   x x
                  x x x
                 x x x x
                x x x x x
               x x x x x x
              x x x x x x x
             x x x x x x x x
            x x x x x x x x x
           x x x x x x x x x x
          x x x x x x x x x x x
         x x x x x x x x x x x x
        x x x x x x x x x x x x x
       x x x x x x x x x x x x x x
      x x x x x x x x x x x x x x x
     x x x x x x x x x x x x x x x x
    x x x x x x x x x x x x x x x x x
   x x x x x x x x x x x x x x x x x x
  x x x x x x x x x x x x x x x x x x x
 x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x x x x x x x x x x x x

baris = 16
```

4.6 Perulangan Tanpa Henti

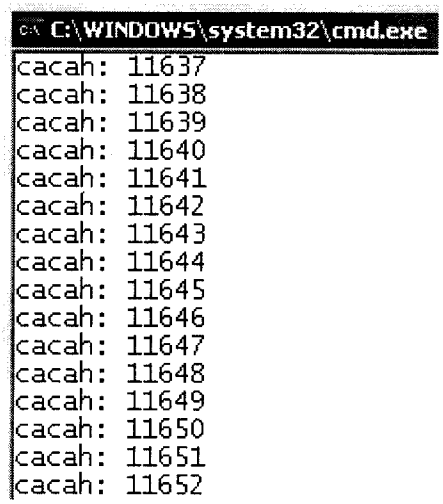
Konsep perulangan-tanpa-henti pada intinya adalah mengisi parameter instruksi *for*, *while* maupun *do-while* dengan logika *true*. Ini disebabkan karena ketiga perulangan ini akan berjalan jika kondisinya bernilai *true*. Jika kondisi perulangan diberi nilai *true* maka komputer akan menganggap bahwa perulangan harus terjadi tanpa syarat, sedangkan jika kondisi perulangan diberi suatu pernyataan relasi maka komputer akan mengevaluasi pernyataan tersebut untuk memutuskan apakah proses berulang akan dikerjakan atau tidak.

Program berikut ini akan menampilkan angka 1, 2, 3, dan seterusnya tanpa henti.

Program 4.27

```
1  public class Program427
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6
7      while (true)
8      {
9        System.out.println("cacah: " + cacah);
10       cacah++;
11     }
12   }
13 }
```


Contoh output program yang sempat tertangkap:



Angka di atas akan terus berjalan tanpa henti sampai di-*stop* secara paksa / manual.

Satu hal yang perlu diingat adalah jika kita memberi perintah berulang tanpa henti seperti di atas, maka instruksi yang ada di bawah blok perulangan tidak akan pernah dilalui. Uniknya, Java mendeteksi kondisi ini sejak proses *compile* berlangsung, lalu proses *compile* dianggap gagal atau error. Program berikut ini akan menunjukkan hal tersebut:

Program 4.28

```
1  public class Program428
2  {
3    public static void main(String[] args)
4    {
5      int cacah = 0;
6
7      while (true)
8      {
9        System.out.println("cacah: " + cacah);
```

```
10          cacah++;
11      }
12
13      System.out.println("Apakah teks ini akan
muncul?");
14    }
15  }
```

Pada saat program ini di-*compile*, Java akan menampilkan pesan error sebagai berikut:

```
C:\WINDOWS\system32\cmd.exe
Program428.java:13: unreachable statement
    System.out.println("Apakah teks ini akan muncul?");
    ^
1 error
```

Pesan di atas menyatakan bahwa instruksi pada baris 13 tidak akan pernah dieksekusi akibat kondisi sebelumnya yang tidak memungkinkan hal itu terjadi. Menarik, bukan?

4.7 Soal Latihan

1. Buatlah program Java untuk mencari Faktor Persekutuan Terbesar (FPB atau GCD, *Greatest Common Divisor*) dari dua buah bilangan integer non-zero!

2. Buatlah program Java untuk mencari Kelipatan Persekutuan Terkecil (KPK) dari dua buah bilangan integer non-zero!

3. Bilangan prima adalah bilangan yang habis dibagi dengan satu dan bilangan itu sendiri. Buatlah program Java untuk menentukan apakah sebuah bilangan integer non-zero sembarang merupakan bilangan prima atau tidak!

4. Buatlah program Java untuk mensimulasikan proses perpangkatan A terhadap B, ditulis dengan notasi A^B, di mana A dan B adalah bilangan integer! Komposisi yang mungkin terjadi adalah: A+B+, A+B-, A-B+ dan A-B-.

5. Buatlah program Java untuk mensimulasikan proses pembagian bilangan A terhadap B, ditulis dengan notasi $^A/_B$, di mana keduanya adalah bilangan integer! Lakukan proses pembagian sampai jumlah digit sebanyak mungkin. Jika Anda menggunakan operator "/" standar, maka Anda akan dibatasi oleh kapasitas memori setiap tipe data.

BAB – 5
VARIABEL ARRAY

Tujuan

1. Pembaca memahami konsep variabel array di Java dan dapat mengimplementasikannya dalam pemrograman.
2. Pembaca mampu membuat program yang menerapkan array satu dimensi dan dua dimensi.
3. Pembaca mampu membuat program untuk mencari data dengan kriteria tertentu.
4. Pembaca mampu membuat program untuk melakukan proses sortir data.
5. Pembaca mampu membuat program untuk memproses array dinamis.
6. Pembaca mampu menggunakan class Vector, Stack, dan Hashtable untuk memanajemen data secara dinamis.

5.1 Pengantar

Program yang cukup kompleks membutuhkan variabel dalam jumlah besar. Kita mungkin saja mendeklarasikan variabel-variabel tersebut satu per satu. Jika kita membutuhkan 5 (lima) variabel bertipe *int*, kita bisa saja mendeklarasikan kelima variabel tersebut dengan cara biasa:

```
int a, b, c, d, e;
```

Di memori kelima variabel tersebut akan memiliki lokasi sebagai berikut:

a → ☐ b→ ☐ c → ☐ d → ☐ e → ☐

Setiap variabel menempati lokasi terpisah satu sama lain. Pengaksesan terhadap variabel-variabel tersebut bisa dilakukan secara langsung dengan mengacu kepada namanya, misalnya untuk menyimpan angka 43 ke dalam variabel c kita bisa memberi instruksi berikut:

```
c = 43;
```

a → ☐ b→ ☐ c → [43] d → ☐ e → ☐

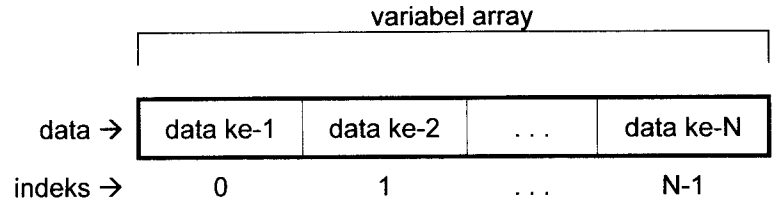
Proses yang sama bisa diterapkan kepada variabel lainnya dengan cara serupa.

Sekarang, bagaimana jika kita ingin mendeklarasikan 100 variabel, apakah kita akan menulisnya satu per-satu, atau kita akan melakukan trik *copy-and-paste*? Cara mana pun yang kita pilih, semuanya akan membuang banyak waktu. Selain itu kita akan kesulitan memilih nama yang *unik*, karena selain harus berbeda satu sama lain, nama variabel hendaknya memiliki makna, bukan sekedar nama. Untuk memenuhi kebutuhan akan 100 variabel tersebut, kita bisa saja mendeklarasikannya dengan cara berikut:


```
int x1, x2, x3, ... x100;
```

Konsep variabel array diberikan untuk menyelesaikan, salah satunya, persoalan deklarasi banyak variabel. Variabel array adalah sekelompok variabel yang memiliki nama sama. Setiap variabel dalam array dikenal dengan sebutan *elemen*. Untuk membedakan satu elemen dengan elemen lainnya digunakan nomor atau index. Elemen pertama diberi index 0, elemen kedua diberi index 1, elemen ketiga diberi index 2, dan seterusnya sampai elemen ke-N diberi index (N-1).

Sebuah variabel array bisa diilustrasikan sebagai berikut:



5.2 Deklarasi Variabel Array

Java menyediakan beberapa cara untuk mendeklarasikan variabel array. Pada prinsipnya cara-cara ini berlaku sama untuk seluruh tipe data termasuk variabel bertipe *class*. Pada bagian ini kita akan fokus kepada tipe data *int* untuk memudahkan pemahaman terhadap materi array. Anda bisa menerapkan cara yang sama untuk tipe data lain seperti *String, double, boolean* dan lain-lain.

Sebuah variabel berjenis `array-of-int` bernama dataku bisa dideklarasikan dengan cara-cara sebagai berikut:

Strategi #1: Deklarasi ala kadarnya

```
int[] dataku;
```

Dengan cara ini variabel dataku dikenal sebagai variabel berjenis array-of-int, akan tetapi banyaknya elemen yang tersimpan di dalamnya belum diketahui. Dengan demikian di dalam program harus ada instruksi untuk menentukan banyaknya elemen dataku, yaitu:

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

Instruksi ini menyatakan bahwa dataku memiliki 5 elemen. Sayangnya kelima elemen tersebut masih kosong. Untuk mengisi data ke dalam elemen-elemen tersebut kita bisa memberi instruksi seperti berikut:

```
dataku[0] = 32;
```

Elemen pertama kita beri data 32 dan elemen lainnya kita biarkan kosong.

Strategi #2: Tentukan Banyak Elemennya

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

Dengan cara ini variabel dataku dikenal sebagai variabel berjenis array-of-int di mana banyaknya elemen yang tersimpan di dalamnya adalah 5. Cara ini lebih cepat satu langkah daripada cara pertama. Sama seperti cara sebelumnya, cara ini belum memasukkan data untuk setiap elemen. Kita bisa menggunakan cara berikut untuk memasukkan data ke dalam elemen array:


```
dataku[0] = 32;
```

Elemen pertama kita beri data 32 dan elemen lainnya kita biarkan kosong.

Strategi #3: Tentukan Data Tiap Elemennya

```
int[] dataku = {32, 51, 26, 97, 3};
```

Dengan cara ini variabel dataku dikenal sebagai variabel berjenis array-of-int di mana banyaknya elemen yang tersimpan di dalamnya adalah 5 dan isi setiap elemen adalah seperti dalam teks di atas. Cara ini memang tidak menyebutkan secara eksplisit jumlah elemen array, akan tetapi berdasarkan banyaknya data yang tertulis di dalam kurung-kurawal, komputer bisa menghitung banyaknya elemen yang dipesan.

Tidak ada aturan yang mengharuskan Anda menggunakan salah satu strategi di atas; Anda bebas memilih berdasarkan kondisi yang dihadapi. Berikut ini kondisi yang umum terjadi:

- Strategi #1 biasa digunakan jika banyaknya elemen dan isi tiap elemen harus ditentukan pada saat *run-time*, mungkin ditentukan melalui keyboard.
- Strategi #2 biasa digunakan jika banyaknya elemen sudah bisa ditentukan sejak awal namun isi tiap elemennya harus dimasukkan oleh user.
- Strategi #3 biasa digunakan jika banyaknya elemen dan isi tiap elemennya sudah diketahui secara pasti, pada umumnya berupa data yang siap diolah.

5.3 Mengakses Variabel Array

Setelah variabel array dideklarasikan secara sempurna, kita bisa mengakses variabel tersebut. Jika kita mendeklarasikan variabel array menggunakan Strategi #1 maupun Strategi #2, variabel array dikatakan sempurna jika kita sudah memberi perintah *new* untuk mengalokasikan memori bagi array tersebut. Tanpa adanya perintah ini variabel array belum bisa digunakan untuk menyimpan data.

Mengakses variabel array meliputi beberapa kegiatan, yaitu menghitung banyaknya elemen array, mengisi nilai ke dalam variabel array, mengambil nilai dari dalam variabel array dan mengubah isi variabel array. Meskipun Java mengijinkan terjadi perubahan banyak elemen array sewaktu-waktu, kami tidak memasukkan topik ini ke dalam sub-bab ini. Hal ini akan dijelaskan pada sub-bab berikutnya mengenai Array Dinamis.

Menghitung Banyaknya Elemen Array

Kelebihan dari sistem variabel array di Java adalah kita bisa menghitung banyaknya elemen suatu variabel array dengan mudah. Perintah *length* digunakan untuk tujuan ini. Harap dibedakan dengan fungsi length() pada data bertipe *String* yang berguna untuk menghitung panjang dari string tersebut, atau dengan kata lain banyaknya karakter yang tersimpan di dalam variabel string tersebut. Perhatikan bahwa *length* pada array tidak diakhiri dengan sepasang tanda kurung.

Instruksi *length* tidak berhubungan dengan data yang tersimpan di dalam sebuah variabel array; perintah ini hanya berguna untuk mengetahui banyaknya elemen dari variabel array tersebut.

Program 5.1

```
1  public class Program51
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7     System.out.println("Banyak elemen dataku: " +
   dataku.length);
8    }
9  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Banyak elemen dataku: 5
```

Mengisi Nilai ke Variabel Array

Setiap elemen dari variabel array bisa diisi secara terpisah sebagaimana variabel biasa, dengan perkecualian bahwa nama dari elemen yang akan diisi data harus diberi nomor indeks.

Program 5.2

```
1  public class Program52
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      dataku[0] = 43;
8      dataku[1] = 62;
9      dataku[2] = 70;
10     dataku[3] = 3;
11     dataku[4] = 37;
```

```
12    }
13  }
```

Jika seluruh elemen array ingin diisi dengan nilai yang sama, kita bisa memanfaatkan proses berulang sebanyak nilai batas yang diperoleh dari perintah *length*.

Program 5.3

```
1  public class Program53
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      for (int i=0; i<dataku.length; i++)
8        dataku[i] = 0;
9    }
10 }
```

Program53 akan mengisi nilai 0 (nol) ke seluruh elemen array dataku. Jika kita tidak ingin mengisikan nilai yang sama ke seluruh elemen array, maka proses berulang, perintah *length* dan fungsi random() bisa digunakan untuk tujuan itu.

Program 5.4

```
1  public class Program54
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      for (int i=0; i<dataku.length; i++)
8        dataku[i] = (int)(Math.random()*100);
9    }
10 }
```


Mengambil Nilai dari Variabel Array

Mengambil nilai dari variabel array dilakukan untuk berbagai tujuan, bisa sekedar untuk dicetak ke layar dan bisa juga ditujukan untuk diproses dengan data lain. Pengambilan data array dilakukan dengan menyebutkan nama variabel array diikuti dengan nomor indeks dari elemen yang akan diambil nilainya.

Program 5.5

```
1  public class Program55
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      for (int i=0; i<dataku.length; i++)
8      {
9        dataku[i] = (int)(Math.random()*100);
10
11       System.out.println("Isi dataku[" + i +
"] --> " + dataku[i]);
12     }
13   }
14 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Isi dataku[0] --> 47
Isi dataku[1] --> 39
Isi dataku[2] --> 76
Isi dataku[3] --> 36
Isi dataku[4] --> 81
```

Mengubah Nilai pada Variabel Array

Isi elemen array bisa diubah dengan cara yang sama dengan mengisi nilai awal ke variabel array. Data yang telah ada digantikan oleh data baru.

Program 5.6

```
1  public class Program56
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      System.out.println("Data mula-mula:");
8      for (int i=0; i<dataku.length; i++)
9      {
10       dataku[i] = (int)(Math.random()*100);
11
12       System.out.println("\tIsi dataku[" + i +
   "] --> " + dataku[i]);
13     }
14
15     System.out.println("\nData sekarang:");
16     for (int i=0; i<dataku.length; i++)
17     {
18       dataku[i] = (int)(Math.random()*100);
19
20       System.out.println("\tIsi dataku[" + i +
   "] --> " + dataku[i]);
21     }
22   }
23 }
```


Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Data mula-mula:
        Isi dataku[0] --> 92
        Isi dataku[1] --> 85
        Isi dataku[2] --> 1
        Isi dataku[3] --> 82
        Isi dataku[4] --> 67

Data sekarang:
        Isi dataku[0] --> 41
        Isi dataku[1] --> 12
        Isi dataku[2] --> 80
        Isi dataku[3] --> 89
        Isi dataku[4] --> 32
```

5.4 Pengurutan Data

Salah satu manfaat penggunaan variabel array adalah kita dapat mengurutkan sejumlah data menurut kriteria tertentu. Saat ini telah banyak metode sortir data yang bisa digunakan, masing-masing memiliki kelebihan dan kekurangan. Ada beberapa parameter yang digunakan untuk menilai baik-buruknya suatu metode sortir, antara lain jumlah langkah yang dilakukan dalam melakukan pengurutan data, banyaknya resource yang digunakan untuk melakukan pengurutan data, banyaknya data yang mampu diurutkan secara cepat, dan lain-lain. Ada metode yang secara konsep mudah dipahami namun lambat jika mengurutkan banyak data, dan ada metode yang secara konsep agak susah dipahami namun dapat mengurutkan banyak data secara cepat. Sampai saat ini upaya untuk menemukan metode sortir yang lebih cepat masih terus dilakukan.

Ascending vs Descending

Orientasi pengurutan data ada dua: *ascending* dan *descending*. Pengurutan secara *ascending* adalah mengatur letak data sedemikian rupa sehingga semakin tinggi indeks suatu elemen maka akan semakin tinggi nilai yang tersimpan di dalamnya dibandingkan dengan nilai pada elemen sebelumnya. Pengurutan secara *descending* adalah mengatur letak data sedemikian rupa sehingga semakin tinggi indeks suatu elemen maka akan semakin rendah nilai yang tersimpan di dalamnya dibandingkan dengan nilai pada elemen sebelumnya.

Form absensi murid merupakan contoh implementasi sortir model *ascending*, di mana nama-nama murid disajikan secara urut berdasarkan nomor induknya dari yang kecil sampai yang besar. Form kelulusan murid merupakan contoh implementasi sortir model *descending*, di mana nama-nama murid disajikan secara urut berdasarkan *ranking* dari nomor besar ke nomor kecil.

Pada prinsipnya kedua model pengurutan data ini memiliki algoritma yang sama. Perbedaan utamanya terletak pada operator relasi yang digunakan saat membandingkan data pada satu elemen dengan data pada elemen yang lain. Lebih jelas lagi tentang ini akan Anda jumpai pada penjelasan selanjutnya.

Selection Sort

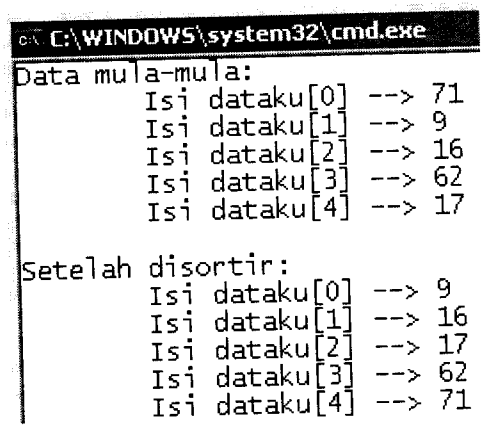
Metode Selection Sort adalah metode yang mudah dipahami dan diimplementasikan namun bekerja sangat lambat jika data yang harus diurutkan sangat banyak, misalnya 10000 data.

Prinsip utama dalam metode ini adalah "memegang" satu elemen lalu membandingkan isinya dengan isi dari elemen-elemen yang terletak sesudahnya. Jika data kedua elemen memiliki urutan yang salah, maka kedua data tersebut akan ditukar tempat sehingga urutan keduanya sesuai dengan kriteria yang telah ditetapkan.

Program 5.7

```
1  public class Program57
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      for (int i=0; i<dataku.length; i++)
8        dataku[i] = (int)(Math.random()*100);
9
10     System.out.println("Data mula-mula:");
11     for (int i=0; i<dataku.length; i++)
12       System.out.println("\tIsi dataku[" + i +
   "] --> " + dataku[i]);
13
14     //.. proses sortir
15     for (int kiri=0; kiri<dataku.length-1;
   kiri++)
16     {
17      for (int kanan=kiri+1; kanan<dataku.length;
   kanan++)
18       {
19         if (dataku[kiri] > dataku[kanan])
20         {
21           int bantu = dataku[kiri];
22           dataku[kiri] = dataku[kanan];
23           dataku[kanan] = bantu;
24         }
25       }
26     }
27     //.. akhir proses sortir
28
29     System.out.println("\nSetelah disortir:");
30     for (int i=0; i<dataku.length; i++)
31       System.out.println("\tIsi dataku[" + i +
   "] --> " + dataku[i]);
32   }
33 }
```

Contoh output program:



Variabel kiri digunakan sebagai indeks bagi elemen sebelah kiri dan variabel kanan digunakan sebagai indeks bagi elemen sebelah kanan. Jika elemen sebelah kiri nilainya lebih besar daripada elemen sebelah kanan, maka kedua data ditukar tempat. Nilai kedua variabel ini tidak pernah sama.

Program di atas akan mengurutkan data secara *ascending*. Jika kita ingin mengurutkan data secara *descending* maka yang perlu diubah hanya baris 19.

Semula tertulis begini:

```
if (dataku[kiri] > dataku[kanan])
```

maka sekarang harus diganti menjadi:

```
if (dataku[kiri] < dataku[kanan])
```

Perbedaannya hanya terletak pada karakter "**>**" yang diganti menjadi "**<**".

Program 5.8

```
1  public class Program58
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      for (int i=0; i<dataku.length; i++)
8        dataku[i] = (int)(Math.random()*100);
9
10     System.out.println("Data mula-mula:");
11     for (int i=0; i<dataku.length; i++)
12       System.out.println("\tIsi dataku[" + i +
"] --> " + dataku[i]);
13
14     //.. proses sortir
15     for (int kiri=0; kiri<dataku.length-1;
kiri++)
16     {
17      for (int kanan=kiri+1; kanan<dataku.length;
kanan++)
18       {
19         if (dataku[kiri] < dataku[kanan])
20         {
21           int bantu = dataku[kiri];
22           dataku[kiri] = dataku[kanan];
23           dataku[kanan] = bantu;
24         }
25       }
26     }
27     //.. akhir proses sortir
28
29     System.out.println("\nSetelah disortir:");
30     for (int i=0; i<dataku.length; i++)
31       System.out.println("\tIsi dataku[" + i +
"] --> " + dataku[i]);
32   }
33 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Data mula-mula:
        Isi dataku[0] --> 13
        Isi dataku[1] --> 11
        Isi dataku[2] --> 68
        Isi dataku[3] --> 58
        Isi dataku[4] --> 17

Setelah disortir:
        Isi dataku[0] --> 68
        Isi dataku[1] --> 58
        Isi dataku[2] --> 17
        Isi dataku[3] --> 13
        Isi dataku[4] --> 11
```

Jika elemen sebelah kiri nilainya lebih kecil daripada elemen sebelah kanan, maka kedua data ditukar tempat.

Istilah "kiri" dan "kanan" digunakan untuk membedakan letak elemen, di mana "kiri" merujuk ke elemen dengan indeks kecil dan "kanan" merujuk ke elemen dengan indeks lebih besar.

Proses sortir dengan metode Selection sort bisa diilustrasikan sebagai berikut, dengan pengurutan secara *ascending* dan data yang diurutkan sesuai dengan data pada Program57.

dataku →	71	9	16	62	17
indeks →	0	1	2	3	4

Proses sortir ditulis dengan tabel penelusuran sebagai berikut:

Variabel		dataku					Keterangan
kiri	kanan	71	9	16	62	17	
0	1	9	71	16	62	17	Elemen 0 dan 1 ditukar (71 ↔ 9)
	2	9	71	16	62	17	
	3	9	71	16	62	17	
	4	9	71	16	62	17	
1	2	9	16	71	62	17	Elemen 1 dan 2 ditukar (71 ↔ 16)
	3	9	16	71	62	17	
	4	9	16	71	62	17	
2	3	9	16	62	71	17	Elemen 2 dan 3 ditukar (71 ↔ 62)
	4	9	16	17	71	62	Elemen 2 dan 4 ditukar (62 ↔ 17)
3	4	9	16	17	62	71	Elemen 3 dan 4 ditukar (71 ↔ 62)

Hasil akhir pengurutan bisa dilihat pada baris terakhir tabel penelusuran di atas, yaitu `9, 16, 17, 62, 71`.

Bubble Sort

Metode Bubble Sort mirip dengan Selection Sort, bedanya adalah kedua perulangan pada Bubble Sort bergerak bersamaan. Jika pada Selection Sort variabel kanan bergerak setelah kiri, maka pada Bubble Sort kedua variabel ini bergerak bersamaan.

Program 5.9

```
1  public class Program59
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[5];
6
7      for (int i=0; i<dataku.length; i++)
8        dataku[i] = (int)(Math.random()*100);
9
10     System.out.println("Data mula-mula:");
11     for (int i=0; i<dataku.length; i++)
12       System.out.println("\tIsi dataku[" + i +
   "] --> " + dataku[i]);
13
14     //.. proses sortir
15     for (int i=dataku.length-1; i>0; i--)
16     {
17       for (int kiri=0; kiri<i; kiri++)
18       {
19         int kanan = kiri + 1;
20
21         if (dataku[kiri] > dataku[kanan])
22         {
23           int bantu = dataku[kiri];
24           dataku[kiri] = dataku[kanan];
25           dataku[kanan] = bantu;
26         }
27       }
28     }
29     //.. akhir proses sortir
30
31     System.out.println("\nSetelah disortir:");
32     for (int i=0; i<dataku.length; i++)
33       System.out.println("\tIsi dataku[" + i +
   "] --> " + dataku[i]);
34   }
35 }
```


Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Data mula-mula:
        Isi dataku[0] --> 71
        Isi dataku[1] --> 85
        Isi dataku[2] --> 25
        Isi dataku[3] --> 88
        Isi dataku[4] --> 67

Setelah disortir:
        Isi dataku[0] --> 25
        Isi dataku[1] --> 67
        Isi dataku[2] --> 71
        Isi dataku[3] --> 85
        Isi dataku[4] --> 88
```

Proses sortir ditulis dengan tabel penelusuran sebagai berikut:

Variabel			dataku					Keterangan
i	kiri	kanan	71	85	25	88	67	
4	0	1	71	85	25	88	67	
	1	2	71	25	85	88	67	Elemen 1 dan 2 ditukar (85 ↔ 25)
	2	3	71	25	85	88	67	
	3	4	71	25	85	67	88	Elemen 3 dan 4 ditukar (88 ↔ 67)
3	0	1	25	71	85	67	88	Elemen 0 dan 1 ditukar (71 ↔ 25)
	1	2	25	71	85	67	88	
	2	3	25	71	67	85	88	Elemen 2 dan 3 ditukar (85 ↔ 67)
2	0	1	25	71	67	85	88	

	1	2	25	67	71	85	88	Elemen 1 dan 2 ditukar (71 ↔ 67)
·	0	1	25	67	71	85	88	

Hasil akhir pengurutan bisa dilihat pada baris terakhir tabel penelusuran di atas, yaitu 25, 67, 71, 85, 88. Variabel "i" digunakan untuk membatasi banyaknya perulangan.

5.5 Pencarian Data

Implementasi lain dari variabel array adalah *searching* atau pencarian data. Pada intinya, *searching* adalah proses membandingkan sebuah data masukan dengan sejumlah data yang tersedia berdasarkan kriteria tertentu. Tujuan pencarian data ada dua: (1) memeriksa ada tidaknya suatu data, dan (2) menampilkan semua data yang sesuai dengan data tertentu. Jika proses pencarian ditujukan untuk memeriksa keberadaan suatu data, maka proses pencarian akan dihentikan begitu data yang cocok pertama kali ditemukan, atau sampai sudah tidak ada data yang bisa dicocokkan. Jika proses pencarian ditujukan untuk menampilkan semua data yang cocok, maka proses pencarian akan dilakukan sampai indeks data terakhir, dan setiap bertemu data yang cocok data itu akan ditampilkan.

Pada bagian ini kita akan membahas tentang metode Sequential Search dan Binary Search, dua metode pencarian data yang umum dikenal.

Sequential Search

Sequential Search adalah metode pencarian data dengan melakukan pencarian data secara urut dari indeks 0 menuju akhir data. Pada setiap nomor indeks tertentu dilakukan pembandingan apakah data pada indeks tersebut sama dengan data yang dicari atau tidak. Metode ini merupakan metode paling sederhana, cukup lakukan pembandingan mulai data pertama sampai data terakhir atau sampai data yang dicari ditemukan.

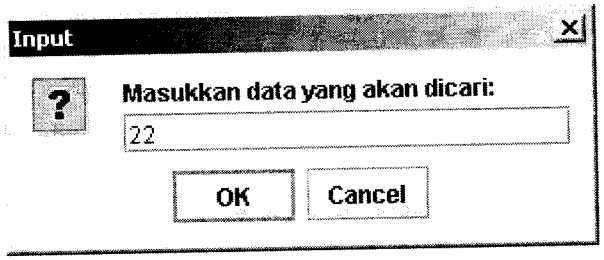
Program 5.10

```
1  import javax.swing.JOptionPane;
2
3  public class Program510
4  {
5    public static void main(String[] args)
6    {
7      int[] dataku = new int[10];
8      int cari;
9
10     System.out.print("Isi database:  ");
11     for (int i=0; i<dataku.length; i++)
12     {
13       dataku[i] = (int)(Math.random()*100);
14       System.out.print(dataku[i] + "  ");
15     }
16
17     String str = JOptionPane.showInputDialog(
"Masukkan data yang akan dicari:");
18     cari = Integer.parseInt(str);
19
20     System.out.println("\n");
21
22     for (int i=0; i<dataku.length; i++)
23     {
24       if (cari == dataku[i])
25       {
26         System.out.println("Data " + cari +
```

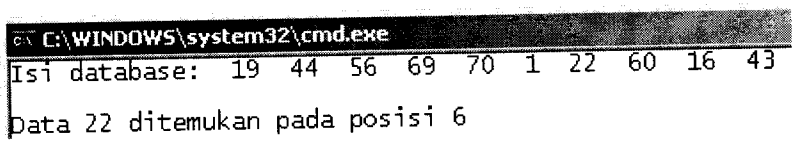
```
" ditemukan pada posisi " + i);
27              break;
28            }
29          }
30        }
31    }
```

Mula-mula komputer akan mengacak 10 bilangan lalu menampilkannya di layar. (asumsi kita memiliki database yang memuat 10 data)

Kemudian komputer akan bertanya data yang akan dicari:



Berdasarkan data yang kita masukkan komputer akan memeriksa keberadaan data tersebut:



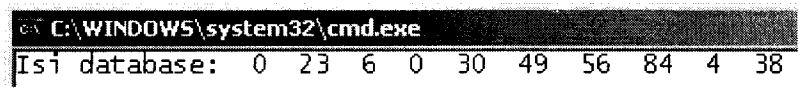
Posisi 6 bermakna sama dengan indeks 6 atau data ke-7.

Pada Program510 baris 27 menulis perintah *break* dengan maksud agar proses pencarian dihentikan begitu data yang dicari ditemukan. Jika perintah *break* dihilangkan maka seluruh data yang cocok akan ditampilkan.

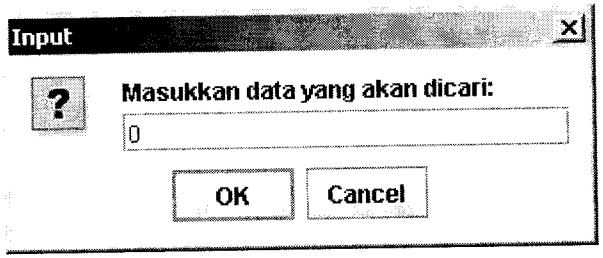
Program 5.11

```
1   import javax.swing.JOptionPane;
2
3   public class Program511
4   {
5     public static void main(String[] args)
6     {
7       int[] dataku = new int[10];
8       int cari;
9
10      System.out.print("Isi database:  ");
11      for (int i=0; i<dataku.length; i++)
12      {
13        dataku[i] = (int)(Math.random()*100);
14        System.out.print(dataku[i] + "  ");
15      }
16
17      String str = JOptionPane.showInputDialog(
"Masukkan data yang akan dicari:");
18      cari = Integer.parseInt(str);
19
20      System.out.println("\n");
21
22      for (int i=0; i<dataku.length; i++)
23      {
24        if (cari == dataku[i])
25          System.out.println("Data " + cari +
" ditemukan pada posisi " + i);
26      }
27    }
28  }
```

Mula-mula komputer akan mengacak 10 bilangan lalu menampilkannya di layar. (asumsi kita memiliki database yang memuat 10 data)



Pada saat komputer meminta masukan data yang akan dicari, masukkan data yang muncul lebih dari satu, yaitu 0.



Hasilnya adalah:

```
C:\WINDOWS\system32\cmd.exe
Isi database:  0  23  6  0  30  49  56  84  4  38

Data 0 ditemukan pada posisi 0
Data 0 ditemukan pada posisi 3
```

Jadi data 0 ditemukan sebagai data ke-1 dan data ke-4.

Binary Search

Metode ini akan melakukan pencarian dengan cara membagi dua daerah pencarian lalu memeriksa apakah data di tengah-tengah sama dengan yang dicari atau tidak. Jika tidak, maka pencarian berikutnya akan dilakukan terhadap setengah data yang kiri atau kanan tergantung pada apakah data tengah tersebut lebih besar atau lebih kecil dari data yang dicari. Jika data tengah lebih besar dari data yang dicari, maka pencarian berikutnya akan difokuskan pada setengah data sebelah kiri. Area pencarian yang baru ini juga akan dipecah menjadi dua bagian dan diperiksa data tengahnya apakah sama dengan data yang dicari atau tidak. Proses "bagi dua dan bandingkan" ini akan terus dilakukan sampai data yang dicari ditemukan atau sampai area yang diperiksa tidak bisa dibagi lagi.

Metode Binary Search hanya bisa dilakukan pada data yang sudah terurut. Jadi sebelum pencarian dengan metode ini dilakukan, pastikan bahwa datanya sudah terurut lebih dulu.

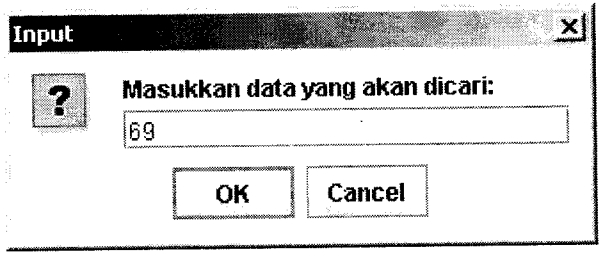
Program 5.12

```
1  import javax.swing.JOptionPane;
2
3  public class Program512
4  {
5    public static void main(String[] args)
6    {
7      int[] dataku = new int[10];
8      int cari;
9
10     System.out.print("Isi database:  ");
11     for (int i=0; i<dataku.length; i++)
12     {
13       dataku[i] = (int)(Math.random()*100);
14       System.out.print(dataku[i] + "  ");
15     }
16
17     String str = JOptionPane.showInputDialog(
   "Masukkan data yang akan dicari:");
18     cari = Integer.parseInt(str);
19
20     System.out.println("\n");
21
22     for (int i=0; i<dataku.length; i++)
23     {
24       if (cari == dataku[i])
25         System.out.println("Data " + cari +
   " ditemukan pada posisi " + i);
26     }
27   }
28 }
```

Mula-mula komputer akan mengacak 10 bilangan lalu menampilkannya di layar. (asumsi kita memiliki database yang memuat 10 data)

```
C:\WINDOWS\system32\cmd.exe
Isi database:
        48  86  69  25  31  53  99  90  69  83

Setelah diurutkan:
        25  31  48  53  69  69  83  86  90  99
```

Kemudian masukkan data yang akan dicari.



Hasilnya adalah:

```
C:\WINDOWS\system32\cmd.exe
Isi database:
        48  86  69  25  31  53  99  90  69  83

Setelah diurutkan:
        25  31  48  53  69  69  83  86  90  99

Data 69 ditemukan pada posisi 5
```

Jadi data 69 ditemukan sebagai data ke-6.

5.6 Manipulasi Data Array

Berikut ini akan dijelaskan beberapa kegiatan yang bisa dilakukan terhadap data variabel array selain kegiatan mengisi data ke dan

mengambil data dari variabel array. Bisa dikatakan bahwa kegiatan yang dimaksud pada bagian ini adalah khusus milik variabel array dan tidak bisa diterapkan pada variabel biasa.

Menyalin Isi Variabel Array

Sebuah array bisa disalin menjadi array lain dengan syarat banyaknya elemen array kedua (*target*) mengikuti banyaknya elemen array pertama (*source*).

Program 5.13

```
1  public class Program513
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[10];
6      int[] target = new int[dataku.length];
7
8      System.out.print("Isi dataku:\n\t");
9      for (int i=0; i<dataku.length; i++)
10     {
11       dataku[i] = (int)(Math.random()*100);
12       System.out.print(dataku[i] + "  ");
13     }
14     System.out.println("\n");
15
16     System.out.print("Isi target mula-mula:
   \n\t");
17     for (int i=0; i<dataku.length; i++)
18       System.out.print(target[i] + "  ");
19     System.out.println("\n");
20
21     //.. proses penyalinan
22     for (int i=0; i<dataku.length; i++)
23       target[i] = dataku[i];
24     //.. akhir proses penyalinan
25
```

```
26      System.out.print("Isi target setelah
    penyalinan:\n\t");
27      for (int i=0; i<dataku.length; i++)
28        System.out.print(target[i] + "  ");
29      System.out.println("\n");
30    }
31  }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Isi dataku:
        10  62  61  10  32  5  10  52  53  23

Isi target mula-mula:
        0  0  0  0  0  0  0  0  0  0

Isi target setelah penyalinan:
        10  62  61  10  32  5  10  52  53  23
```

Menghapus Elemen Array

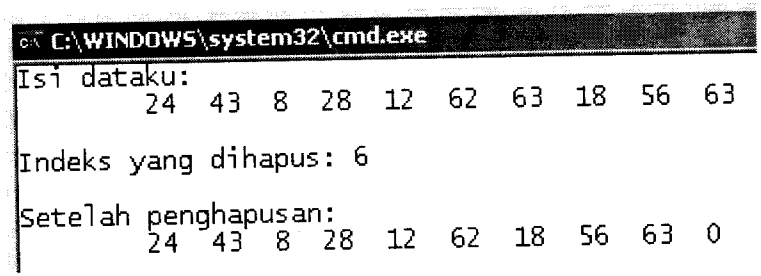
Istilah menghapus elemen array bisa memiliki dua makna yaitu (1) menggeser seluruh elemen pada indeks sesudah indeks terhapus satu lokasi ke atas/kiri lalu mengisi nilai 0 pada indeks terakhir, dan (2) mengisi nilai *null* pada indeks yang dinyatakan dihapus. Data *null* di Java mengandung arti bahwa suatu variabel belum memiliki lokasi pasti di memori sehingga ke dalam variabel itu kita tidak bisa menyimpan data.

Berikut ini akan kita lihat dua program yang mensimulasikan proses menghapus elemen array. Program514 mensimulasikan penghapusan elemen array dengan cara menggeser seluruh elemen array setelah indeks yang dinyatakan dihapus satu lokasi ke atas/kiri lalu mengisi nilai 0 pada elemen terakhir.

Program 5.14

```
1   public class Program514
2   {
3     public static void main(String[] args)
4     {
5       int[] dataku = new int[10];
6       int dihapus = (int)(Math.random()*
    dataku.length);
7
8       System.out.print("Isi dataku:\n\t");
9       for (int i=0; i<dataku.length; i++)
10      {
11        dataku[i] = (int)(Math.random()*100);
12        System.out.print(dataku[i] + "  ");
13      }
14      System.out.println("\n");
15
16      System.out.println("Indeks yang dihapus: " +
    dihapus + "\n");
17
18      //.. proses penghapusan
19      for (int i=dihapus+1; i<dataku.length; i++)
20        dataku[i-1] = dataku[i];
21
22      dataku[dataku.length-1] = 0;
23      //.. akhir proses penghapusan
24
25      System.out.print("Setelah
    penghapusan:\n\t");
26      for (int i=0; i<dataku.length; i++)
27        System.out.print(dataku[i] + "  ");
28      System.out.println("\n");
29    }
30  }
```

Contoh output program:



Lokasi yang akan dihapus diacak oleh komputer, lihat baris 6.

Menyisipkan Elemen Array

Menyisipkan data baru ke dalam variabel array akan berdampak ukuran array bertambah satu, dengan asumsi hanya satu data yang bisa disisipkan pada satu saat. Anda bisa berkreasi untuk mengembangkan konsep ini agar bisa menambahkan beberapa data sekaligus ke dalam variabel array.

Pada dasarnya kegiatan menyisipkan data baru ke dalam variabel array dilakukan dengan menggeser seluruh elemen yang terletak sesudah indeks terpilih satu posisi ke kanan/belakang, lalu pada indeks tersebut dimasukkan data baru. Karena Java tidak menyediakan fitur untuk proses ini, maka kita akan melakukan trik sebagai berikut:

- Salin seluruh isi array *source* ke dalam array *target*.
- Definisi ulang array *source* dengan banyak elemen ditambah satu.
- Saling isi array *target* ke dalam array *source* satu per-satu, dan jika indeks yang di-*scan* sama dengan indeks sisipan maka masukkan data baru ke dalam array *source*; selain itu masukkan data *target* ke dalam *source*.

Program 5.15

```
1  public class Program515
2  {
3    public static void main(String[] args)
4    {
5      int[] dataku = new int[10];
6      int[] target = new int[dataku.length];
7      int sisipan = (int)(Math.random()*
   dataku.length);
8      int dataBaru = (int)(Math.random()*100);
9
10     System.out.print("Isi dataku:\n\t");
11     for (int i=0; i<dataku.length; i++)
12     {
13       dataku[i] = (int)(Math.random()*100);
14       System.out.print(dataku[i] + "  ");
15     }
16     System.out.println("\n");
17
18     System.out.println("Indeks sisipan: " +
   sisipan);
19     System.out.println("  Data sisipan: " +
   dataBaru);
20     System.out.println();
21
22     //.. proses penyisipan
23     for (int i=0; i<dataku.length; i++)
24       target[i] = dataku[i];
25
26     dataku = new int[target.length+1];
27
28     for (int i=0, j=0; i<dataku.length; i++)
29     {
30       if (i == sisipan)
31         dataku[i] = dataBaru;
32       else
33       {
34         dataku[i] = target[j];
35         j++;
```

```
36          }
37        }
38        //.. akhir proses penyisipan
39
40        System.out.print("Setelah penyisipan:\n\t");
41        for (int i=0; i<dataku.length; i++)
42          System.out.print(dataku[i] + "  ");
43        System.out.println("\n");
44      }
45  }
```

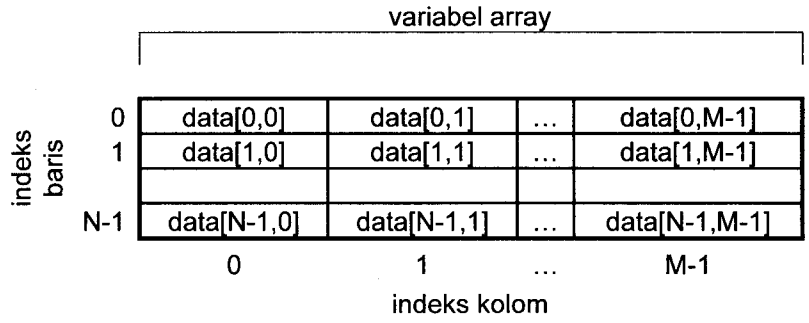
Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Isi dataku:
        44  94  9  42  73  79  26  36  21  54

Indeks sisipan: 4
  Data sisipan: 38

Setelah penyisipan:
        44  94  9  42  38  73  79  26  36  21  54
```

5.7 Array Dua Dimensi

Array dua dimensi merupakan perluasan dari array satu dimensi yang telah kita bahas panjang lebar di atas. Bentuk dari array dua dimensi adalah seperti papan catur atau seperti matrik, ada deretan data yang disebut baris dan ada yang disebut kolom. Disebut array dua dimensi karena untuk mencapai suatu elemen array dibutuhkan dua bilangan index: satu untuk menyatakan baris dan satu lagi untuk menyatakan kolom. Array dua dimensi yang hanya memiliki satu kolom atau satu baris saja akan memiliki bentuk seperti array satu dimensi.

Sebuah variabel array dua dimensi dengan ordo NxM --banyaknya baris adalah N dan banyaknya kolom adalah M-- bisa diilustrasikan sebagai berikut:



Deklarasi Variabel Array Dua Dimensi

Sebuah variabel array dua dimensi bisa dideklarasikan dengan salah satu cara berikut:

Cara #1: Deklarasi Sederhana

```
int[][] dataku;

dataku = new int[5][3];
```

Variabel dataku didefinisikan bertipe array-of-int dengan ordo 5 baris x 3 kolom secara tidak langsung di dalam badan program.

Cara #2: Tentukan Ordonya

```
int[][] dataku = new int[5][3];
```

Variabel dataku didefinisikan bertipe array-of-int dengan ordo 5 baris x 3 kolom secara langsung.

Cara #3: Tentukan Nilai Tiap Elemennya

```
int[][] dataku = { {8, 4, 3},
                   {9, 5, 6},
                   {3, 2, 7},
                   {6, 8, 0},
                   {4, 2, 5} };
```

Variabel dataku didefinisikan bertipe array-of-int dengan ordo 5 baris x 3 kolom secara tidak langsung berdasarkan banyaknya data yang dimasukkan. Model penulisan seperti di atas adalah *style* masing-masing orang, tidak ada keharusan menulis seperti itu. Yang penting adalah tepat dalam meletakkan kurung kurawal buka dan tutupnya.

Konsep array dua dimensi di Java sedikit berbeda dengan array di bahasa pemrograman lain. Di Java, sebuah array dua dimensi tidak harus memiliki kolom yang sama banyak untuk setiap barisnya. Bisa saja baris pertama memiliki 3 kolom sedangkan pada baris keduanya terdapat 1 kolom. Contoh deklarasinya adalah sebagai berikut:

```
int[][] dataku = { {8, 4, 3},
                   {9, 6},
                   {3, 6, 2, 7},
                   {6, 8, 0},
                   {4, 5} };
```

Hal ini memberi kesempatan kepada kita untuk mengatur sendiri memori yang akan digunakan sebagai variabel array agar

pemakaiannya efisien. Seringkali array berbentuk matrik menghabiskan memori secara percuma karena tidak semua lokasi digunakan dalam pengolahan data.

Perintah "length" pada Array Dua Dimensi

Dengan sifatnya yang memiliki baris dan kolom, perintah *length* pada array dua dimensi memiliki dua makna: (1) menyatakan banyaknya baris dari array, dan (2) menyatakan banyaknya kolom untuk baris tertentu.

Jika diketahui deklarasi sebagai berikut:

```
int[][] dataku = { {8, 4, 3},
                   {9, 6},
                   {3, 6, 2, 7},
                   {6, 8, 0},
                   {4, 5} };
```

maka:

- Pernyataan `dataku.length` bermakna banyaknya baris dari variabel dataku, pada contoh ini nilainya adalah 5.
- Pernyataan `dataku[1].length` bermakna banyaknya kolom dari variabel dataku pada baris kedua, pada contoh ini nilainya adalah 2, pernyataan `dataku[2].length` bermakna banyaknya kolom dari variabel dataku pada baris ketiga, pada contoh ini nilainya adalah 4, dan seterusnya.

Mengakses Data Array Dua Dimensi

Pengaksesan elemen dari variabel array dua dimensi bisa dilakukan seperti array satu dimensi dengan tambahan sebuah angka indeks lagi.

Program berikut ini mencontohkan aktivitas-aktivitas yang bisa terjadi pada variabel array dua dimensi. Program ini akan membentuk sebuah struktur segitiga Pascal di mana bilangan tepi kiri dan kanan adalah 1 dan bilangan di tengah diperoleh dari penjumlahan dua bilangan di atasnya.

Program 5.16

```
1  public class Program516
2  {
3    public static void main(String[] args)
4    {
5      int max = (int)(1 + Math.random()*10);
6      int[][] segitiga = new int[max][];
7
8      System.out.println("Banyak baris: " + max +
"\n");
9      System.out.println("Segitiga Pascal:");
10
11     //.. menentukan banyaknya kolom tiap barisnya
12     for (int i=0; i<segitiga.length; i++)
13       segitiga[i] = new int[i+1];
14
15     //.. menentukan isi tiap elemen
16     for (int i=0; i<segitiga.length; i++)
17     {
18       for (int j=0; j<segitiga[i].length; j++)
19       {
20         if ( (j == 0) || (i == j) )
21           segitiga[i][j] = 1;
22         else
23           segitiga[i][j] = segitiga[i-1][j-1] +
24                            segitiga[i-1][j];
25       }
26     }
27
28     //.. mencetak isi segitiga
29     for (int i=0; i<segitiga.length; i++)
30     {
31       for (int j=0; j<segitiga[i].length; j++)
```



```
32         System.out.print(segitiga[i][j] + "  ");
33
34       System.out.println();
35     }
36   }
37 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Banyak baris: 4

Segitiga Pascal:
1
1  1
1  2  1
1  3  3  1
```

Penjumlahan Matrik

Operasi matrik merupakan contoh kasus yang sering digunakan untuk menjelaskan konsep array dua dimensi, misalnya penjumlahan, perkalian, transpose dan inverse. Penjumlahan matrik terhadap dua buah matrik bisa dilakukan jika kedua matrik memiliki ordo yang sama. Matrik hasil penjumlahan adalah sebuah matrik dengan ordo sama di mana setiap elemennya pada baris dan kolom tertentu merupakan hasil penjumlahan dari setiap elemen matrik 1 dan matrik 2 pada baris dan kolom yang bersesuaian.

Misalkan ada dua matrik A dan B masing-masing berordo NxM:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & & & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \dots & & & \dots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix}$$

Maka matrik hasil penjumlahan keduanya, sebut saja matrik C, bisa diilustrasikan sebagai berikut:

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \dots & & & \dots \\ c_{n1} & c_{n2} & \dots & c_{nm} \end{bmatrix}$$

di mana:

- $c_{11} = a_{11} + b_{11}$
- $c_{21} = a_{21} + b_{21}$
- $c_{n1} = a_{n1} + b_{n1}$

Secara umum:

$$c_{ij} = a_{ij} + b_{ij}$$

Berikut ini adalah contoh program untuk penjumlahan matrik. Tentu saja ini juga berlaku untuk pengurangan matrik, karena pada hakikatnya pengurangan adalah penjumlahan dengan bilangan negatif. Pada program ini data matrik 1 dan matrik 2 maupun ordo keduanya di-*generate* secara acak.

Program 5.17

```
1  public class Program517
2  {
3    public static void main(String[] args)
4    {
5      int baris = (int)(1 + Math.random()*10);
6      int kolom = (int)(1 + Math.random()*10);
7
8      int[][] matrik1 = new int[baris][kolom];
9      int[][] matrik2 = new int[baris][kolom];
10     int[][] matrik3 = new int[baris][kolom];
11
12     //.. inisialisasi kedua matrik
13     for (int i=0; i<baris; i++)
14     {
15       for (int j=0; j<kolom; j++)
16       {
17        matrik1[i][j]= (int)(Math.random()*19-9);
18        matrik2[i][j]= (int)(Math.random()*19-9);
19       }
20     }
21
22     //.. proses penjumlahan matrik
23     for (int i=0; i<baris; i++)
24       for (int j=0; j<kolom; j++)
25         matrik3[i][j] = matrik1[i][j] +
   matrik2[i][j];
26
27     //.. mencetak data matrik 1
28     System.out.println("Matrik 1:");
29     for (int i=0; i<matrik1.length; i++)
30     {
31       System.out.print("\t");
32       for (int j=0; j<matrik1[i].length; j++)
33       {
34         if (matrik1[i][j] >= 0)
35           System.out.print(" ");
36
37         System.out.print(matrik1[i][j] + "  ");
```

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

    //.. mencetak data matrik 2
    System.out.println("Matrik 2:");
    for (int i=0; i<matrik2.length; i++)
    {
      System.out.print("\t");
      for (int j=0; j<matrik2[i].length; j++)
      {
        if (matrik2[i][j] >= 0)
          System.out.print(" ");

        System.out.print(matrik2[i][j] + "  ");
      }
      System.out.println();
    }
    System.out.println();

    //.. mencetak matrik hasil penjumlahan
    System.out.println("Matrik 3:");
    for (int i=0; i<matrik3.length; i++)
    {
      System.out.print("\t");
      for (int j=0; j<matrik3[i].length; j++)
      {
        if ( (matrik3[i][j] > -10) &&
(matrik3[i][j] < 0) )
          System.out.print(" ");
        else
        if ( (matrik3[i][j] >= 0) &&
(matrik3[i][j] < 10) )
          System.out.print("  ");
        else
        if (matrik3[i][j] >= 10)
          System.out.print(" ");

```



```
        System.out.print(matrik3[i][j] + "  ");
      }
      System.out.println();
    }
    System.out.println();
  }
}
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Matrik 1:
         3  -4   5   2  -2   0   1   6   3   9
        -1  -8  -8   8   4  -3  -1  -2  -2   9
         1   2   0  -2  -3   6  -3   5  -2  -1
        -6  -5  -5   8   2  -8  -2  -4   5   9

Matrik 2:
        -3  -5   7   3  -8   5  -2   9  -2  -6
         2   4   0   6   3  -2   8   5   9   1
         1  -4  -8  -1   0  -5  -4   0   5   0
         3  -7  -5   9   0   5   3   1  -3   0

Matrik 3:
         0   -9   12    5  -10    5   -1   15    1    3
         1   -4   -8   14    7   -5    7    3    7   10
         2   -2   -8   -3   -3    1   -7    5    3   -1
        -3  -12  -10   17    2   -3    1   -3    2    9
```

Perkalian Matrik

Pada perkalian matrik A dan matrik B, jika ordo A adalah NxM, maka ordo matrik B haruslah MxP. Dengan kata lain ordo baris matrik kedua harus sama dengan ordo kolom matrik pertama. Matrik hasil perkalian akan memiliki ordo NxP, ordo barisnya mengikuti baris matrik pertama dan ordo kolomnya mengikuti kolom matrik kedua.

Misalkan ada dua matrik A berordo NxM dan matrik B berordo MxP:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & & & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & & & \dots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{bmatrix}$$

Maka matrik hasil perkalian keduanya, sebut saja matrik C, bisa diilustrasikan sebagai berikut:

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \dots & & & \dots \\ c_{n1} & c_{n2} & \dots & c_{np} \end{bmatrix}$$

di mana:

- $c_{11} = (a_{11} \times b_{11}) + (a_{12} \times b_{21}) + (a_{13} \times b_{31}) + \dots + (a_{1m} \times b_{m1})$
- $c_{12} = (a_{11} \times b_{12}) + (a_{12} \times b_{22}) + (a_{13} \times b_{32}) + \dots + (a_{1m} \times b_{m2})$
- $c_{21} = (a_{21} \times b_{12}) + (a_{22} \times b_{22}) + (a_{23} \times b_{32}) + \dots + (a_{2m} \times b_{m2})$

Secara umum:

$$c_{ij} = (a_{i1} \times b_{1j}) + (a_{i2} \times b_{2j}) + (a_{i3} \times b_{3j}) + \dots + (a_{im} \times b_{mj})$$

Berikut ini adalah contoh program untuk perkalian matrik. Pada program ini ordo dan data matrik 1 di-*generate* secara acak. Ordo baris matrik 2 disesuaikan dengan kolom matrik 1. Ordo kolom dan data matrik 2 di-*generate* secara acak.

Program 5.18

```
 1  public class Program518
 2  {
 3    public static void main(String[] args)
 4    {
 5      int baris1 = (int)(1 + Math.random()*10);
 6      int kolom1 = (int)(1 + Math.random()*10);
 7
 8      int baris2 = kolom1;
 9      int kolom2 = (int)(1 + Math.random()*10);
10
11      int[][] matrik1 = new int[baris1][kolom1];
12      int[][] matrik2 = new int[baris2][kolom2];
13      int[][] matrik3 = new int[baris1][kolom2];
14
15      //.. inisialisasi matrik 1
16      for (int i=0; i<baris1; i++)
17        for (int j=0; j<kolom1; j++)
18         matrik1[i][j]= (int)(Math.random()*19-9);
19
20      //.. inisialisasi matrik 2
21      For (int i=0; i<baris2; i++)
22        for (int j=0; j<kolom2; j++)
23         matrik2[i][j]= (int)(Math.random()*19-9);
24
25      //.. proses perkalian matrik
26      for (int i=0; i<baris1; i++)
27        for (int j=0; j<kolom2; j++)
28        {
29          matrik3[i][j] = 0;
30
31          for (int k=0; k<kolom1; k++)
32            matrik3[i][j] = matrik3[i][j] +
(matrik1[i][k] * matrik2[k][j]);
```

```
    }

  //.. mencetak data matrik 1
  System.out.println("Matrik 1:");
  for (int i=0; i<matrik1.length; i++)
  {
    System.out.print("\t");
    for (int j=0; j<matrik1[i].length; j++)
    {
      if (matrik1[i][j] >= 0)
        System.out.print(" ");

      System.out.print(matrik1[i][j] + "  ");
    }
    System.out.println();
  }
  System.out.println();

  //.. mencetak data matrik 2
  System.out.println("Matrik 2:");
  for (int i=0; i<matrik2.length; i++)
  {
    System.out.print("\t");
    for (int j=0; j<matrik2[i].length; j++)
    {
      if (matrik2[i][j] >= 0)
        System.out.print(" ");

      System.out.print(matrik2[i][j] + "  ");
    }
    System.out.println();
  }
  System.out.println();

  //.. mencetak matrik hasil penjumlahan
  System.out.println("Matrik 3:");
  for (int i=0; i<matrik3.length; i++)
  {
    System.out.print("\t");
```



```
    for (int j=0; j<matrik3[i].length; j++)
    {
      if ( (matrik3[i][j] > -10) &&
(matrik3[i][j] < 0) )
        System.out.print(" ");
      else
      if ( (matrik3[i][j] >= 0) &&
(matrik3[i][j] < 10) )
        System.out.print("  ");
      else
      if (matrik3[i][j] >= 10)
        System.out.print(" ");

      System.out.print(matrik3[i][j] + "  ");
    }
    System.out.println();
  }
  System.out.println();
 }
}
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Matrik 1:
        3  -4   5   2  -2   0   1   6   3   9
       -1  -8  -8   8   4  -3  -1  -2  -2   9
        1   2   0  -2  -3   6  -3   5  -2  -1
       -6  -5  -5   8   2  -8  -2  -4   5   9

Matrik 2:
       -3  -5   7   3  -8   5  -2   9  -2  -6
        2   4   0   6   3  -2   8   5   9   1
        1  -4  -8  -1   0  -5  -4   0   5   0
        3  -7  -5   9   0   5   3   1  -3   0

Matrik 3:
        0   -9   12    5  -10    5   -1   15    1    3
        1   -4   -8   14    7   -5    7    3    7   10
        2   -2   -8   -3   -3    1   -7    5    3   -1
       -3  -12  -10   17    2   -3    1   -3    2    9
```

Matrik Transpose

Matrik transpose adalah matrik yang memiliki ordo terbalik dari matrik asal. Jika matrik asal A memiliki ordo NxM, maka matrik transpose A^T akan memiliki ordo MxN.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & & & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$$

maka matrik transpose-nya adalah:

$$A^T = \begin{bmatrix} a^t_{11} & a^t_{21} & \dots & a^t_{n1} \\ a^t_{12} & a^t_{22} & \dots & a^t_{n2} \\ \dots & & & \dots \\ a^t_{1m} & a^t_{2m} & \dots & a^t_{nm} \end{bmatrix}$$

dimana:

- $a^t_{11} = a_{11}$
- $a^t_{12} = a_{21}$
- $a^t_{21} = a_{12}$

Secara umum:

$$a^t_{ij} = a_{ji}$$

Berikut ini adalah contoh program untuk transpose matrik. Pada program ini baik ordo maupun data matrik di-*generate* secara acak.

Program 5.19

```
1  public class Program519
2  {
3    public static void main(String[] args)
4    {
5      int baris = (int)(1 + Math.random()*10);
6      int kolom = (int)(1 + Math.random()*10);
7
8      int[][] matrik1 = new int[baris][kolom];
9      int[][] matrik2 = new int[kolom][baris];
10
11     //.. inisialisasi matrik asal
12     for (int i=0; i<baris; i++)
13       for (int j=0; j<kolom; j++)
14        matrik1[i][j]= (int)(Math.random()*19-9);
15
16     //.. proses transpose matrik
17     for (int i=0; i<baris; i++)
18       for (int j=0; j<kolom; j++)
19         matrik2[j][i] = matrik1[i][j];
20
21     //.. mencetak data matrik asal
22     System.out.println("Matrik asal:");
23     for (int i=0; i<matrik1.length; i++)
24     {
25       System.out.print("\t");
26       for (int j=0; j<matrik1[i].length; j++)
27       {
28         if (matrik1[i][j] >= 0)
29           System.out.print(" ");
30
31         System.out.print(matrik1[i][j] + "  ");
32       }
33       System.out.println();
34     }
```

```
35      System.out.println();
36
37      //.. mencetak matrik hasil transpose
38      System.out.println("Matrik transpose:");
39      for (int i=0; i<matrik2.length; i++)
40      {
41        System.out.print("\t");
42        for (int j=0; j<matrik2[i].length; j++)
43        {
44          if (matrik2[i][j] >= 0)
45            System.out.print(" ");
46
47          System.out.print(matrik2[i][j] + "  ");
48        }
49        System.out.println();
50      }
51      System.out.println();
52    }
53  }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Matrik asal:
         0   0   6  -5
         0   5  -3  -7
        -8   7   6   0
        -1   5   3   3
         7   0  -5   0

Matrik transpose:
         0   0  -8  -1   7
         0   5   7   5   0
         6  -3   6   3  -5
        -5  -7   0   3   0
```


5.8 Array Dinamis

Array Dinamis adalah konsep yang memungkinkan terjadinya *growable array*, yaitu variabel array yang banyak elemennya bisa bertambah atau berkurang pada saat program sedang berjalan. Pada bahasa pemrograman tradisional hal ini dikenal dengan sebutan, meski tidak 100% tepat, *pointer* atau *linked-list*. Pada saat kebutuhan akan variabel meningkat, program akan mengalokasi sejumlah memori untuk menyimpan data, dan pada saat ada variabel yang sudah tidak terpakai lagi maka program akan membuangnya dari memori sehingga tidak terjadi penumpukan "sampah data".

Array Dinamis Semu

Istilah array dinamis semu kami pilih karena pada dasarnya implementasi dari array dinamis ini hanyalah *trik* dengan memanipulasi banyaknya elemen suatu array. Strategi yang digunakan adalah:

- Buat variabel array cadangan dengan ukuran seperti variabel array asli.
- Salin seluruh data dari variabel array asli ke variabel array cadangan.
- Definisi ulang variabel array asli dengan banyak elemen sesuai permintaan.
- Lakukan penyalinan data dari array cadangan ke array asli yang baru.
- Jika ukuran array asli yang baru lebih besar daripada ukuran array cadangan, sisa elemen pada array asli diberi nilai 0.

Program berikut ini akan mencontohkan bagaimana kita bisa membuat array dinamis dengan trik tersebut.

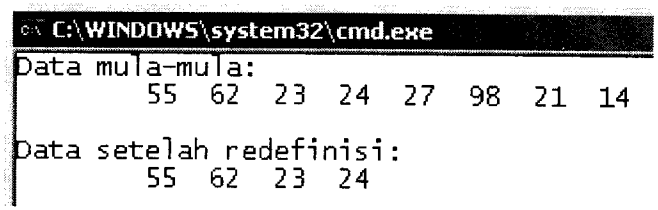
Program 5.20

```
1  public class Program520
2  {
3    public static void main(String[] args)
4    {
5      int jumElemen = (int)(1 + Math.random()*10);
6      int[] dataAsli = new int[jumElemen];
7
8      //.. inisialisasi data array asli
9      for (int i=0; i<dataAsli.length; i++)
10       dataAsli[i] = (int)(Math.random()*100);
11
12     //.. menampilkan isi array asli
13     System.out.print("Data mula-mula:\n\t");
14     for (int i=0; i<dataAsli.length; i++)
15       System.out.print(dataAsli[i] + "  ");
16     System.out.println('\n');
17
18     //.. menyalin data asli ke array cadangan
19     int[] cadangan = new int[dataAsli.length];
20
21     for (int i=0; i<dataAsli.length; i++)
22       cadangan[i] = dataAsli[i];
23
24     //.. mengubah ukuran array
25     jumElemen = (int)(1 + Math.random()*10);
26     dataAsli = new int[jumElemen];
27
28     //.. menyalin data cadangan ke array asli
29     for (int i=0; i<cadangan.length; i++)
30       if (i < dataAsli.length)
31         dataAsli[i] = cadangan[i];
32
33     if (dataAsli.length > cadangan.length)
34       for (int i=cadangan.length;
   i<dataAsli.length; i++)
35         dataAsli[i] = 0;
36
37     //.. menampilkan isi array asli
```

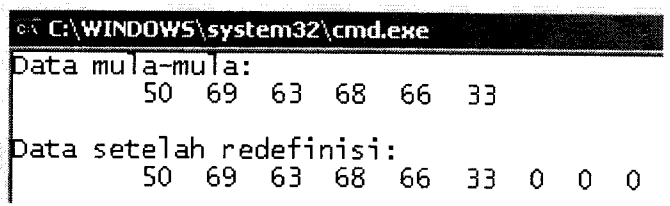


```
38     System.out.print("Data setelah redefinisi:
   \n\t");
39     for (int i=0; i<dataAsli.length; i++)
40       System.out.print(dataAsli[i] + "  ");
41     System.out.println('\n');
42   }
43 }
```

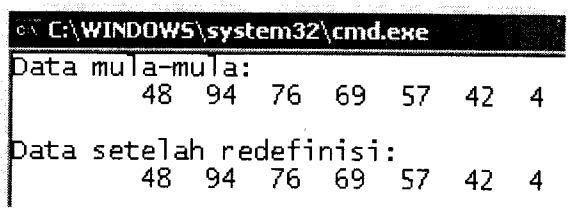
Contoh output #1: ukuran baru lebih kecil daripada ukuran lama.



Contoh output #2: ukuran baru lebih besar daripada ukuran lama.



Contoh output #3: ukuran baru sama dengan ukuran lama.



Baris 30 menjamin bahwa data yang disalin memiliki indeks lebih kecil daripada ukuran array yang baru. Jika ini tidak dilakukan,

dikuatirkan akan terjadi *Exception* akibat pengaksesan array pada indeks yang di luar range.

Baris 33 s/d 35 merupakan antisipasi terhadap kemungkinan ukuran array yang baru lebih besar daripada ukuran array semula, dimana jika itu terjadi maka seluruh elemen sisa akan diberi nilai nol. Terhadap variabel array bertipe *char* anda bisa memberi nilai kosong berupa karakter *spacebar* atau spasi yaitu ASCII 32. Untuk variabel array bertipe *String* anda bisa memberi nilai kosong berupa *empty-string*.

Class Vector

Class *Vector* merupakan salah satu fitur yang disediakan Java untuk implementasi array dinamis. Data yang disimpan di dalam sebuah *Vector* bertipe *Object*, yaitu class spesial milik Java yang mewakili data apa saja. Ini berarti kita bisa menyimpan data dari sembarang tipe ke dalam objek *Vector*, termasuk juga kita bisa menyimpan data berjenis *Vector* ke dalam *Vector* yang lain, begitu seterusnya sampai tidak terhingga.

Sebuah vektor bisa dibayangkan sebagai sebuah daftar / list yang berisi data per-baris dimana setiap data memiliki tipe sembarang, bahkan bisa jadi data tersebut bertipe vektor yang lain. Kita bisa mengakses data pada sembarang posisi.

Sebuah objek *Vector* dideklarasikan dengan cara sebagai berikut:

```
Vector vektorku = new Vector();
```

Berikut ini diberikan daftar method atau fungsi yang disediakan oleh class *Vector* yang sering digunakan.

Method Class Vector

boolean	add(E element) Menambahkan element sebagai data terakhir objek vektor.
void	add(int index, E element) Menambahkan element ke dalam vektor pada index tertentu.
void	addElement(E obj) Menambahkan elemen baru, obj, sebagai data terakhir objek vektor.
int	capacity() Mengembalikan nilai yang menyatakan kapasitas objek vektor.
void	clear() Menghapus seluruh elemen pada objek vektor.
Object	clone() Mengembalikan objek yang merupakan *clone* atau duplikat dari objek vektor.
boolean	contains(Object obj) Mengembalikan nilai *true* jika objek obj ada di dalam objek vektor.
void	copyInto(Object[] arr) Menyalin seluruh elemen pada objek vektor ke dalam variabel array arr yang bertipe array-of-object.

E elementAt(int index)

Mengembalikan nilai yang menyatakan elemen dari objek vektor yang berada pada index tertentu.

Enumeration elements()

Mengembalikan nilai berupa daftar seluruh komponen yang ada di dalam objek vektor dalam bentuk *enumerasi*.

boolean equals(Object obj)

Mengembalikan nilai *true* jika objek obj sama dengan objek vektor.

E firstElement()

Mengembalikan nilai yang menyatakan elemen pertama dari objek vektor.

int indexOf(Object obj)

Mengembalikan nomor index yang pertama kali datanya sama dengan obj di mana proses pencarian dimulai dari indeks 0 dan bergerak menuju index terakhir. Jika tidak ada yang cocok maka nilai yang dikembalikan adalah -1.

int indexOf(Object obj, int idx)

Mengembalikan nomor index yang pertama kali datanya sama dengan obj di mana proses pencarian dimulai dari indeks idx dan bergerak menuju index terakhir. Jika tidak ada yang cocok maka nilai yang dikembalikan adalah -1.

void insertElementAt(E obj, int index)

Menyisipkan element baru obj pada index tertentu.

boolean isEmpty()

Mengembalikan nilai *true* jika objek vektor dalam keadaan tidak memiliki data.

E lastElement()

Mengembalikan nilai yang menyatakan elemen terakhir dari objek vektor.

int lastIndexOf(Object obj)

Mengambalikan nilai yang menyatakan index terakhir yang datanya sesuai dengan objek obj di mana pencarian dilakukan mulai dari index terakhir dan bergerak menuju index 0. Jika tidak ada yang cocok maka nilai yang dikembalikan adalah -1.

int lastIndexOf(Object obj, int idx)

Mengambalikan nilai yang menyatakan index terakhir yang datanya sesuai dengan objek obj di mana pencarian dilakukan mulai dari index idx dan bergerak menuju index 0. Jika tidak ada yang cocok maka nilai yang dikembalikan adalah -1.

E remove(int index)

Membuang elemen yang berada pada nomor index tertentu.

Boolean remove(Object obj)

Membuang elemen yang pertama kali cocok datanya sama dengan objek obj. Jika data yang dimaksud tidak ada, isi vektor tidak mengalami perubahan.

Void removeAllElements()

Membuang seluruh elemen vektor dan me-*reset* ukuran vektor ke nilai 0.

void	removeElementAt(int index) Membuang elemen vektor yang berada pada index tertentu.
protected void	removeRange(int fromIndex, int toIndex) Membuang elemen vektor yang nomor indeksnya berada antara fromIndex sampai dengan toIndex-1. Elemen dengan nomor index sama dengan toIndex tidak ikut dihapus.
void	setElementAt(E obj, int index) Mengubah elemen pada nomor index tertentu dengan data obj.
int	size() Mengembalikan nilai yang menyatakan banyak elemen dalam objek vektor.
Object[]	toArray() Mengembalikan data array yang berisi seluruh elemen vektor dalam urutan sebenarnya.

Program berikut ini akan mencontohkan pemakaian sejumlah perintah dalam class *Vector*. Untuk perintah yang tidak sempat dibahas, kami persilahkan Anda mencobanya sendiri.

Program 5.21

```
1  import java.util.Vector;
2
3  public class Program521
4  {
5    public static void main(String[] args)
6    {
7      Vector vectorku = new Vector();
8
```



```
9      for (int i=0; i<5; i++)
10     {
11       int state = (int)(Math.random()*3);
12       int index = (int)(Math.random()*
vectorku.size());
13       int value = (int)(Math.random()*100);
14
15       if ( (state == 0) && !vectorku.isEmpty() )
16       {
17         System.out.println("Action:
removeElementAt(" + index + ")");
18         vectorku.removeElementAt(index);
19       }
20       else
21       {
22         System.out.println("Action:    add("    +
index
+ "," + value + ")");
23         vectorku.add(index,value);
24       }
25
26       System.out.print("Vector: ");
27       for (int j=0; j<vectorku.size(); j++)
28         System.out.print(vectorku.elementAt(j) +
"  ");
29       System.out.println('\n');
30     }
31   }
32 }
```

Program ini akan melakukan 5 proses acak, bisa berupa penambahan elemen baru atau penghapusan elemen yang sudah ada; ini ditentukan oleh variabel state yang akan bernilai acak antara 0 s/d 2. Sengaja kami atur agar peluang penghapusan lebih kecil daripada peluang penambahan elemen dengan pertimbangan agar peluang ada data yang ditampilkan ke layar lebih besar.

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Action: add(0,25)
Vector: 25

Action: removeElementAt(0)
Vector:

Action: add(0,11)
Vector: 11

Action: add(0,11)
Vector: 11  11

Action: add(1,71)
Vector: 11  71  11
```

Jika terjadi penambahan elemen baru pada index yang sudah berisi data, maka data lama akan digeser ke belakang.

Berikut ini diberikan contoh objek vektor berada di dalam objek vektor yang lain.

Program 5.22

```
1  import java.util.Vector;
2
3  public class Program522
4  {
5    public static void main(String[] args)
6    {
7      Vector vector1 = new Vector();
8      Vector vector2 = new Vector();
9
10     vector1.add((int)(Math.random()*100));
11     vector1.add((int)(Math.random()*100));
12     vector1.add((int)(Math.random()*100));
13
14     vector2.add((int)(Math.random()*100));
```



```
15     vector2.add((int)(Math.random()*100));
16     vector2.add(vector1);
17     vector2.add((int)(Math.random()*100));
18
19     //.. mencetak vector1
20     System.out.print("Vector1:  ");
21     for (int j=0; j<vector1.size(); j++)
22       System.out.print(vector1.elementAt(j) +
"  ");
23     System.out.println('\n');
24
25     //.. mencetak vector2
26     System.out.print("Vector2:  ");
27     for (int j=0; j<vector2.size(); j++)
28       System.out.print(vector2.elementAt(j) +
"  ");
29     System.out.println('\n');
30   }
31 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Vector1:  8  77  20

Vector2:  64  20  [8, 77, 20]  5
```

Baris 16 menyatakan bahwa data vector1 dimasukkan sebagai elemen dari vector2.

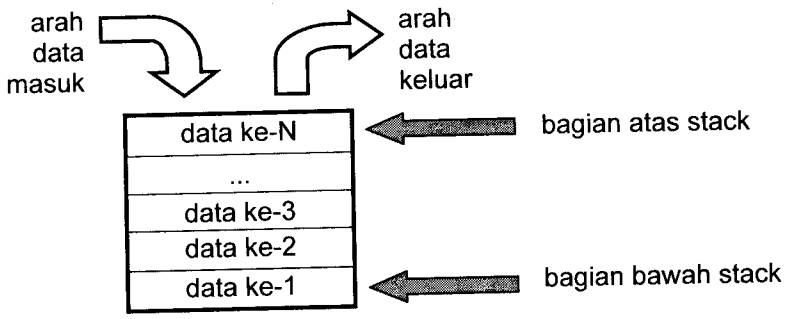
Class Stack

Class *Stack* mengimplementasikan array dinamis dengan sistem Last In First Out (LIFO). Sebuah stack bisa dibayangkan sebagai tabung penyimpanan bola tennis, satu ujung terbuka dan ujung lainnya tertutup. Pemasukan dan pengeluaran bola tennis hanya bisa dilakukan terhadap ujung tabung yang terbuka. Bola yang

dimasukkan ke dalam tabung terakhir kali akan diambil pertama kali, dan bola yang dimasukkan pertama kali akan dikeluarkan terakhir.

Konsep stack digunakan oleh komputer dalam manajemen sub-proses. Jika terjadi pemanggilan terhadap suatu sub-proses, maka alur saat ini akan ditinggalkan lalu dialihkan ke sub-proses yang dipanggil tersebut. Namun sebelum alur saat ini ditinggalkan, komputer akan mencatat terlebih dahulu lokasi yang akan ditinggalkan. Tujuannya adalah jika sub-proses yang akan didatangi sudah selesai dikerjakan, maka lokasi yang akan menjadi tujuan balik bisa diambil dari dalam stack. Tidak jadi masalah jika kemudian di dalam sub-proses yang didatangi terjadi pemanggilan terhadap sub-proses lainnya, demikian terus tanpa batas, komputer tetap akan mencatat lokasi yang akan ditinggalkan ke dalam stack sebagai data terakhir. Nantinya komputer akan kembali ke lokasi semula dengan tepat.

Stack dapat diilustrasikan dengan gambar berikut:



Sebuah objek *Stack* dideklarasikan dengan cara sebagai berikut:

```
Stack stackku = new Stack();
```


Berikut ini diberikan daftar method atau fungsi yang disediakan oleh class *Stack* yang sering digunakan.

Method Class Stack

boolean empty()

Mengembalikan nilai *true* jika objek stack dalam keadaan kosong / tidak memiliki data.

E peek()

Menginformasikan data yang berada pada sisi teratas dari stack tanpa mengambilnya dari stack.

E pop()

Mengembalikan nilai berupa data yang saat ini berada pada puncak stack dan mengambilnya dari stack.

E push(E item)

Memasukkan data baru ke dalam stack. Otomatis data ini akan menempati puncak stack.

int search(Object obj)

Memeriksa keberadaan objek obj di dalam stack. Jika ditemukan, fungsi ini akan mengembalikan nilai yang menunjukkan jaraknya dari puncak stack, di mana puncak stack bernilai 1. Jika tidak ditemukan, fungsi ini akan mengembalikan nilai -1.

Program berikut ini adalah contoh implementasi class *Stack*.

Program 5.23

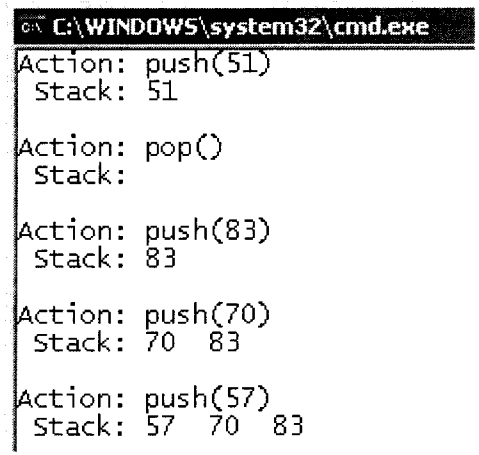
```
1  import java.util.Stack;
2
3  public class Program523
4  {
5    public static void main(String[] args)
6    {
7      Stack stackku = new Stack();
8      Stack cadangan = new Stack();
9
10     for (int i=0; i<5; i++)
11     {
12       int state = (int)(Math.random()*3);
13       int value = (int)(Math.random()*100);
14
15       if ( (state == 0) && !stackku.empty() )
16       {
17         System.out.println("Action: pop()");
18         stackku.pop();
19       }
20       else
21       {
22         System.out.println("Action: push(" +
   value + ")");
23         stackku.push(value);
24       }
25
26       System.out.print(" Stack: ");
27
28       while (!stackku.empty())
29       {
30         Integer data = (Integer)stackku.pop();
31         value = data.intValue();
32         cadangan.push(value);
33
34         System.out.print(value + "  ");
35       }
```



```
36
37       while (!cadangan.empty())
38         stackku.push(cadangan.pop());
39
40       System.out.println('\n');
41     }
42   }
43 }
```

Program ini akan melakukan 5 proses acak, bisa berupa pemasukan elemen baru ke dalam stack atau pengambilan data dari stack; ini ditentukan oleh variabel state yang akan bernilai acak antara 0 s/d 2.

Contoh output program:



Pada saat data stack dicetak ke layar, berhubung pencetakan itu berarti mengambil data dari stack asli, maka digunakan stack bantuan yang akan menampung isi stack asli selama proses pencetakan berlangsung (baris 30 s/d 32). Setelah pencetakan selesai, isi stack bantuan dikembalikan ke stack asli (baris 37 s/d 38).

Sekarang adalah contoh program di mana ada elemen *Stack* yang bertipe *Vector*.

Program 5.24

```
1  import java.util.Vector;
2  import java.util.Stack;
3
4  public class Program524
5  {
6    public static void main(String[] args)
7    {
8      Vector vector = new Vector();
9      Stack stack = new Stack();
10
11     vector.add((int)(Math.random()*100));
12     vector.add((int)(Math.random()*100));
13     vector.add((int)(Math.random()*100));
14     vector.add((int)(Math.random()*100));
15     vector.add((int)(Math.random()*100));
16
17     stack.push((int)(Math.random()*100));
18     stack.push(vector);
19     stack.push((int)(Math.random()*100));
20     stack.push((int)(Math.random()*100));
21
22     //.. mencetak isi vector
23     System.out.print("Vector:  ");
24     for (int j=0; j<vector.size(); j++)
25       System.out.print(vector.elementAt(j) +
"  ");
26     System.out.println('\n');
27
28     //.. mencetak isi stack
29     while (!stack.empty())
30     {
31       Object obj = stack.pop();
32
33       if (obj instanceof Vector)
```



```
34       {
35         Vector v = (Vector)obj;
36
37         System.out.print("[");
38         for (int j=0; j<v.size(); j++)
39           System.out.print(v.elementAt(j) +
"  ");
40         System.out.print("\b\b]  ");
41       }
42       else
43         System.out.print(((Integer)obj).
intValue() + "  ");
44     }
45   }
46 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Vector:  54  43  38  65  36

70  0  [54  43  38  65  36]  65
```

Class Hashtable

Class *Hashtable* mengimplementasikan konsep array dinamis di mana pada setiap baris data tersimpan dua informasi: key dan value. Key adalah data unik yang digunakan untuk mengidentifikasi value. Hashtable adalah contoh penerapan metode *indexing*. Proses pencarian data pada hashtable akan berlangsung cepat karena item yang dijadikan acuan adalah key, bukan data itu sendiri. Value pada sebuah *Hashtable* bisa berupa *Hashtable* yang lain atau *Vector* atau sembarang objek.

Hashtable dapat diilustrasikan dengan gambar berikut:

key-N	value-N
...	...
key-3	value-3
key-2	value-2
key-1	value-1

Sepintas *Hashtable* mirip dengan *Stack*; bedanya adalah *Hashtable* memiliki key dan data di dalamnya bisa diakses pada sembarang posisi berdasarkan key-nya, sedangkan *Stack* hanya mengijinkan input dan output data pada puncak stack.

Sebuah objek *Hashtable* dideklarasikan dengan cara sebagai berikut:

```
Hashtable hashku = new Hashtable();
```

Berikut ini diberikan daftar method atau fungsi yang disediakan oleh class *Hashtable* yang sering digunakan.

Method Class Hashtable

void clear()

Mengosongkan seluruh key yang ada di dalam hashtable.

Object clone()

Menghasilkan objek baru duplikat dari objek hashtable ini.

boolean contains(Object obj)

Menghasilkan nilai *true* jika ada key yang terasosiasi dengan value obj.

boolean containsKey(Object obj)

Menghasilkan nilai *true* jika parameter obj merupakan salah satu key dalam objek hashtable.

boolean containsValue(Object obj)

Menghasilkan nilai *true* jika di dalam objek hashtable ini ada satu atau lebih key yang terasosiasi dengan value obj.

Enumeration elements()

Mengambil seluruh elemen hashtable dalam bentuk *Enumeration*.

V get(Object obj)

Mengembalikan value yang key-nya adalah obj atau *null* jika tidak ada data yang cocok.

boolean isEmpty()

Menghasilkan nilai *true* jika tidak ada key yang terasosiasi dengan value tertentu.

V put(K key, V value)

Memetakan key dengan value.

V remove(Object obj)

Membuang key obj dan value yang terasosiasi dengannya dari dalam objek hashtable.

int size()

Mengembalikan nilai yang menyatakan banyaknya key di dalam objek hashtable.

Program berikut ini adalah contoh implementasi class *Hashtable*.

Program 5.25

```
1  import java.util.Hashtable;
2  import java.util.Enumeration;
3
4  public class Program525
5  {
6    public static void main(String[] args)
7    {
8      Hashtable hash = new Hashtable();
9      Enumeration en = null;
10     int i = 0;
11
12     for ( ; i<10; i++)
13       hash.put(i, (int)(Math.random()*100));
14
15     en = hash.elements();
16
17     System.out.println("Isi hashtable:");
18     while (en.hasMoreElements())
19       System.out.println("\tKey: " + --i +
   "\tValue: " + en.nextElement());
20
21     int keyCari = (int)(Math.random()*10);
22     System.out.println("\nDicari: " + keyCari);
23     System.out.println(" Value: " +
   hash.get(keyCari));
24   }
25 }
```


Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
Isi hashtable:
        Key: 9  Value: 91
        Key: 8  Value: 51
        Key: 7  Value: 10
        Key: 6  Value: 96
        Key: 5  Value: 48
        Key: 4  Value: 82
        Key: 3  Value: 50
        Key: 2  Value: 77
        Key: 1  Value: 98
        Key: 0  Value: 2

Dicari: 7
 Value: 10
```

Enumeration akan menyimpan elemen dengan urutan terbalik.

5.9 Soal Latihan

1. Implementasikan deret bilangan Fibonacci menggunakan konsep variabel array!

2. Matrik Inverse adalah sebuah matrik yang jika dikalikan dengan matrik asal akan menghasilkan matrik identitas. Perhitungan untuk memperoleh matrik identitas mudah dilakukan terhadap matrik berordo 2x2 namun akan semakin sulit untuk ordo lebih besar. Gauss menawarkan metode untuk mencari matrik inverse dengan cara yang bisa diterapkan menggunakan komputer. Implementasikan metode Gauss tersebut untuk mencari matrik inverse!

3. Implementasikan proses konversi bilangan dari sistem Desimal ke dalam Sistem Biner, Oktal dan Hexadesimal dengan konsep variabel array!

BAB – 6
SUB PROGRAM

Tujuan

1. Pembaca memahami pengertian sub program, baik yang berjenis *procedure* maupun yang berjenis *function*.
2. Pembaca mampu membuat program sederhana yang menerapkan pemakaian sub program.
3. Pembaca mampu membuat program menggunakan sub program dengan pengiriman parameter biasa.
4. Pembaca mampu membuat program menggunakan sub program dengan pengiriman parameter berupa data array.
5. Pembaca mampu membuat program yang menerapkan *function overloading*.
6. Pembaca mampu membuat program yang menerapkan *recursive function*.

6.1 Pengantar

Persoalan besar akan lebih mudah diselesaikan jika kita mampu membaginya menjadi beberapa persoalan kecil dan menyelesaikannya secara parsial. Manfaatnya adalah kita bisa melokalisir persoalan lalu bisa melakukan *debugging* lebih cepat. Setiap persoalan kecil ini akan melakukan kegiatan tertentu atau spesifik.

Suatu proses spesifik, seperti membuat kop surat, biasanya memiliki alur yang sudah pasti. Urutan penyelesaian prosesnya pun sudah tertentu dan jarang berubah. Analogi dengan dunia pemrograman, sebuah proses tertentu mungkin saja sudah memiliki algoritma dan susunan instruksi yang pasti. Jika proses ini diperlukan di beberapa lokasi kode program yang berbeda dan digunakan berkali-kali, maka penulisan ulang kode untuk proses tersebut menjadi tidak praktis, tidak perduli berapa banyak baris kode yang diperlukan. Selain itu perubahan algoritma dan susunan kode program akan menjadi masalah besar karena anda harus rela menelusuri satu persatu baris program yang akan diperbaiki.

Sub program adalah sekumpulan instruksi yang akan melakukan proses spesifik dalam rangka menyelesaikan persoalan pemrograman secara utuh. Beberapa sub program akan saling berinteraksi membentuk sebuah program besar. Jika pada saat program ini dijalankan lalu terjadi *run-time error*, programmer akan bisa memperkirakan lokasi kesalahan berdasarkan jenis kesalahan yang terjadi.

Sebuah sub program didefinisikan dengan syntax sebagai berikut:

```
tipe_data nama_sub_program(daftar_parameter)
{
   … instruksi pembentuk sub program …
}
```


Contoh:

```
double waktuTempuh(int v, int s)
{
   return s/v;
}
```

Penamaan sub program memiliki aturan yaitu:

- Diawali dengan huruf kecil.
- Kata kedua, ketiga dan seterusnya diawali dengan huruf kapital.
- Antar kata tidak dipisahkan dengan *underscore* (_) melainkan ditulis sambung.

Sebuah sub program boleh saja memiliki lebih dari satu instruksi. Pemanggilan terhadap sub program sama artinya dengan kita menulis instruksi di dalamnya pada lokasi di mana kita memanggil sub program tersebut. Pemanggilan berulang kali terhadap suatu sub program di beberapa lokasi sama artinya dengan kita menulis seluruh instruksi sub program secara berulang-ulang di beberapa lokasi tempat kita memanggilnya.

Jika kita tidak menggunakan sub program, lalu di beberapa lokasi yang instruksi sejenis kita perlu melakukan perubahan kode program, maka bisa diketahui betapa repotnya kita harus menelusuri beberapa lokasi yang kodenya harus diganti. Pada program yang besar, kegiatan seperti ini sangat melelahkan.

Lain halnya jika kita menggunakan sub program; perubahan kode cukup dilakukan di dalam sub program, maka secara otomatis perubahan itu akan berdampak pada seluruh lokasi yang melakukan pemanggilan terhadap sub program tersebut. Hemat waktu, hemat tenaga, dan sangat efisien.

Secara garis besar sub program dibagi dua kelompok: *procedure* dan *function*.

6.2 Procedure

Procedure adalah jenis sub program yang jika dipanggil akan melakukan suatu proses tertentu tetapi tidak hasil proses di dalam dirinya sendiri.

Procedure didefinisikan dengan syntax sebagai berikut:

```
void namaProsedur(parameter)
{
   ...instruksi yang dikerjakan...
}
```

Kata kunci *void* digunakan untuk mengawali definisi prosedur.

Berikut ini akan diberikan contoh program yang melibatkan prosedur.

Program 6.1

```
 1  public class Program61
 2  {
 3    void informasi()
 4    {
 5      System.out.println("Contoh prosedur");
 6    }
 7
 8    public static void main(String[] args)
 9    {
10      new Program61().informasi();
11    }
12  }
```


Output program:

```
C:\WINDOWS\system32\cmd.exe
Contoh prosedur
```

Program61 merupakan contoh sederhana implementasi prosedur. Kalimat "Contoh prosedur" ditampilkan melalui prosedur informasi(). Baris 10 menunjukkan cara memanggil prosedur tersebut.

Program 6.2

```
 1  public class Program62
 2  {
 3    static void informasi()
 4    {
 5      System.out.println("Contoh prosedur");
 6    }
 7
 8    public static void main(String[] args)
 9    {
10      informasi();
11    }
12  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Contoh prosedur
```

Program62 memperlihatkan cara lain untuk memanggil prosedur, yaitu tulis nama prosedur tersebut tanpa didahului nama class tempatnya berada. Jika cara ini yang kita pilih maka kita harus hati-hati bahwa sebuah sub program yang akan dipanggil dari lokasi *static* haruslah berjenis *static* juga.

Pada Program61 prosedur informasi() tidak didefinisikan *static* karena dia tidak dipanggil secara langsung melainkan melalui

class tempatnya berada. Pada Program62 prosedur informasi() akan dipanggil secara langsung dari fungsi main() yang berjenis *static*, maka dia juga harus didefinisikan static. Jika ini tidak kita lakukan, maka compiler Java akan melaporkan suatu error sebagaimana ditunjukkan pada program berikut.

Program 6.3

```
1  public class Program63
2  {
3    void informasi()
4    {
5      System.out.println("Contoh prosedur");
6    }
7
8    public static void main(String[] args)
9    {
10     informasi();
11   }
12 }
```

Pada saat *compile* Java akan melaporkan error sebagai berikut:

```
C:\WINDOWS\system32\cmd.exe
Program63.java:10: non-static method informasi() cannot be referenced from a sta
tic context
    informasi();
    ^
1 error
```

Pesan tersebut menginformasikan bahwa method yang non-static tidak bisa dipanggil dari lokasi yang static.

Sebuah prosedur boleh berisi lebih dari satu instruksi tergantung kebutuhan. Selain itu sebuah prosedur boleh dipanggil berulang kali sesuai keperluan.

Program 6.4

```
1  public class Program64
2  {
```



```
3    static void informasi()
4    {
5      System.out.println("---------------");
6      System.out.println("Contoh prosedur");
7      System.out.println("---------------");
8    }
9
10   public static void main(String[] args)
11   {
12     informasi();
13     informasi();
14     informasi();
15   }
16 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
---------------
Contoh prosedur
---------------
---------------
Contoh prosedur
---------------
---------------
Contoh prosedur
---------------
```

Mungkin Anda kurang suka garis model "-". Anda bisa melakukan perubahan model garis cukup pada prosedur informasi(). Anda tidak perlu melakukan perubahan garis pada baris 12 s/d 14, karena secara otomatis perubahan yang terjadi di dalam prosedur informasi() akan berpengaruh kepada ketiga baris tersebut.

Program 6.5

```
1  public class Program65
2  {
3    static void informasi()
4    {
5      System.out.println("~~~~~~~~~~~~~~~");
```

```
6       System.out.println("Contoh prosedur");
7       System.out.println("+++++++++++++++");
8     }
9
10    public static void main(String[] args)
11    {
12      informasi();
13      informasi();
14      informasi();
15    }
16  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
~~~~~~~~~~~~~~~~
Contoh prosedur
+++++++++++++++
~~~~~~~~~~~~~~~~
Contoh prosedur
+++++++++++++++
~~~~~~~~~~~~~~~~
Contoh prosedur
+++++++++++++++
```

6.3 Function

Function adalah sub program yang setelah dipanggil akan menyimpan nilai tertentu di dalam dirinya. Function ditandai dengan adanya instruksi *return x* untuk mengembalikan alur proses kepada pemanggilnya, di mana "x" adalah nilai yang akan disimpan ke dalam fungsi.

Function didefinisikan dengan syntax sebagai berikut:


```
tipe namaFunction(parameter)
{
   …instruksi yang dikerjakan…

   return nilai;
}
```

Tipe adalah sembarang tipe data, bisa tipe primitif dan bisa tipe class. *Nilai* yang dikembalikan sebagai hasil fungsi harus bertipe sama dengan tipe fungsi. *Nilai* itu sendiri bisa berupa data, ekspresi maupun variabel.

Program 6.6

```
1   public class Program66
2   {
3     static String informasi()
4     {
5       String str = "~~~~~~~~~~~~~~~~";
6       str += "\nContoh function";
7       str += "\n+++++++++++++++";
8
9       return str;
10    }
11
12    public static void main(String[] args)
13    {
14      System.out.println(informasi());
15    }
16  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
~~~~~~~~~~~~~~~~
Contoh function
+++++++++++++++
```

Sebuah *function* bisa dipanggil sebagai argumen sub program lain maupun bisa diberikan kepada variabel melalui *assignment statement*.

Program 6.7

```
1  public class Program67
2  {
3    static String informasi()
4    {
5      String str = "~~~~~~~~~~~~~~~";
6      str += "\nContoh function";
7      str += "\n+++++++++++++++";
8
9      return str;
10   }
11
12   public static void main(String[] args)
13   {
14     String x = informasi();
15
16     System.out.println(x);
17   }
18 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
~~~~~~~~~~~~~~~
Contoh function
+++++++++++++++
```

Pada baris 14 kita memanggil fungsi informasi() dan menampung hasil prosesnya pada variabel "x". Akhirnya kita mencetak isi variabel "x" tersebut ke layar. Hasilnya sama dengan mencetak fungsi secara langsung seperti pada program sebelumnya.

Untuk function yang mengembalikan nilai numerik, berikut ini contohnya:

Program 6.8

```
1  public class Program68
2  {
3    static double phi()
4    {
5      return 22.0/7;
6    }
7
8    public static void main(String[] args)
9    {
10     System.out.println("Nilai PHI: " + phi());
11   }
12 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
Nilai PHI: 3.142857142857143
```

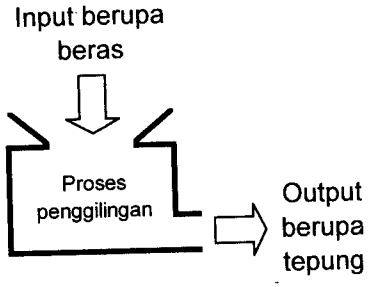
6.4 Parameter Fungsi

Dalam banyak hal sebuah sub program membutuhkan data yang akan diolah untuk di-*supply* dari luar. Data masukan ini disebut *parameter* atau *argumen*. Perlu diketahui sejak saat ini kita akan menyebut fungsi untuk sub-program, baik berjenis *procedure* maupun *function*.

Parameter fungsi pada dasarnya sama dengan variabel biasa, hanya saja nilainya diberikan pada saat suatu fungsi dipanggil. *Parameter formal* adalah nama parameter yang tertulis di dalam definisi fungsi. *Parameter actual* adalah nama variabel yang dikirimkan sebagai data masukan bagi sebuah fungsi. Nama variabel yang dikirim sebagai parameter tidak harus sama dengan nama parameter itu di dalam definisi fungsi. Yang harus sama

adalah tipe datanya. Tidak ada batasan mengenai jumlah dan tipe parameter.

Konsep parameter bisa digambarkan sebagai berikut:



Mesin penggilingan beras yang tidak diberi masukan berupa beras tidak akan menghasilkan output berupa tepung.

Program berikut ini mencontohkan pemakaian parameter fungsi.

Program 6.9

```
1  public class Program69
2  {
3    static double luas(double radius)
4    {
5      return 22.0/7 * radius * radius;
6    }
7
8    public static void main(String[] args)
9    {
10     double jariJari = 5.3;
11     double luasLingkaran = luas(jariJari);
12
13     System.out.println("radius: " + jariJari);
14     System.out.println("  luas: " +
   luasLingkaran);
15   }
16 }
```


Output program:

```
C:\WINDOWS\system32\cmd.exe
radius: 5.3
  luas: 88.28285714285713
```

Pada baris 3 parameter formalnya bernama radius, tetapi pada baris 11 parameter actualnya adalah jariJari. Beda nama tidak masalah asalkan tipe datanya sama.

Parameter Tipe Primitif

Parameter fungsi bisa berupa tipe primitif, misalnya *int* atau *boolean.*

Program 6.10

```
1  public class Program610
2  {
3    static double luas(double p, double l)
4    {
5      return p * l;
6    }
7
8    public static void main(String[] args)
9    {
10     double panjang = Math.random() * 1000;
11     double lebar = Math.random() * 1000;
12     double luasKotak = luas(panjang,lebar);
13
14     System.out.println("panjang: " + panjang);
15     System.out.println("  lebar: " + lebar);
16     System.out.println("   luas: " + luasKotak);
17   }
18 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
panjang: 775.4480348156931
  lebar: 723.4782204526125
   luas: 561019.7642819331
```

Parameter Tipe Array

Fungsi dengan parameter bertipe array biasa digunakan untuk mengolah data array, misalnya menampilkan sembarang data array ke layar. Kehadiran perintah *length* pada array memberi kemudahan bagi kita untuk mengolah seluruh data array.

Program 6.11

```
1  public class Program611
2  {
3    //
4    // Inisialisasi data array
5    //
6    static int[] initialize()
7    {
8      int[] bantu = new int[5];
9
10     for (int i=0; i<bantu.length; i++)
11       bantu[i] = (int)(Math.random()*10);
12
13     return bantu;
14   }
15
16   //
17   // Menampilkan data array
18   //
19   static void display(int[] x, boolean z)
20   {
21     for (int i=0; i<x.length; i++)
22       System.out.print(x[i] + "  ");
```



```
23     System.out.println();
24
25     if (z)
26       System.out.println();
27   }
28
29   //
30   // Main program
31   //
32   public static void main(String[] args)
33   {
34     int[] data1 = initialize();
35     int[] data2 = initialize();
36     int[] data3 = initialize();
37
38     display(data1, true);
39     display(data2, true);
40     display(data3, false);
41   }
42 }
```

Contoh output program:

```
C:\WINDOWS\system32\cmd.exe
0  0  3  3  9

7  5  8  6  9

5  7  0  6  6
```

Kita bisa bekerja dengan data array dengan lebih nyaman karena beberapa proses inti sudah dibuatkan fungsinya, tinggal panggil, beres.

Parameter Tipe Class

Java adalah bahasa pemrograman yang murni berbasis objek (Object Oriented). Java menyediakan banyak tipe data model class. Tipe data class bisa dikirim sebagai parameter. Tipe data class akan didiskusikan lebih mendalam pada buku yang sama jilid 2 ketika membahas tentang Object Oriented Programming (OOP). Ciri utama tipe data class adalah namanya diawali dengan huruf kapital.

Program 6.12

```
1  public class Program612
2  {
3    //
4    // Menampilkan pesan tertentu
5    //
6    static void showMessage(String str)
7    {
8      System.out.println(str + "\n");
9    }
10
11   //
12   // Main program
13   //
14   public static void main(String[] args)
15   {
16     showMessage("Sub program");
17     showMessage("Parameter");
18     showMessage("Procedure");
19     showMessage("Function");
20   }
21 }
```


Output program:

```
C:\WINDOWS\system32\cmd.exe
Sub program

Parameter

Procedure

Function
```

6.5 Function Overloading

Ketika kita akan membuat fungsi untuk menghitung pembagian dua bilangan, kita harus memutuskan apakah tipe parameter yang akan diolah bertipe integer atau pecahan. Kita tahu bahwa proses pembagian keduanya sama saja; yang berbeda adalah hasil bagi yang diperoleh akan mengandung angka pecahan atau tidak.

Untuk mengakomodasi berbagai tipe data yang dibutuhkan pada sebuah fungsi, cara yang bisa dilakukan adalah membuat banyak fungsi dengan nama yang sama namun seluruhnya berbeda dalam parameter:

- Tipe data parameter.
- Banyaknya parameter.
- Urutan tipe data parameter.

Kondisi di mana terdapat beberapa fungsi memiliki nama sama namun berbeda di tingkat parameter disebut dengan *function-overloading*.

Berikut ini diberikan dua contoh program untuk mengilustrasikan konsep function overloading tersebut. Program pertama adalah

pembagian dua bilangan dan program kedua adalah pembuatan garis.

Program 6.13

```
1  public class Program613
2  {
3    //
4    // Pembagian bilangan bulat
5    //
6    static int pembagian(int a, int b)
7    {
8      return a/b;
9    }
10
11   //
12   // Pembagian bilangan pecahan
13   //
14   static double pembagian(double a, double b)
15   {
16     return a/b;
17   }
18
19   //
20   // Main program
21   //
22   public static void main(String[] args)
23   {
24     System.out.println("5 / 2 ==> " +
     pembagian(5,2));
25     System.out.println("5.0 / 2 ==> " +
     pembagian(5.0,2));
26   }
27 }
```


Output program:

```
C:\WINDOWS\system32\cmd.exe
5 / 2 ==> 2
5.0 / 2 ==> 2.5
```

Secara cerdik Java akan menentukan fungsi pembagian() mana yang akan dimasuki terkait dengan tipe data yang dikirim.

Program kedua: menampilkan garis ke layar.

Program 6.14

```
1  public class Program614
2  {
3    //.. garis model #1
4    static void buatGaris()
5    {
6      System.out.println("---------------");
7    }
8
9    //.. garis model #2
10   static void buatGaris(int n)
11   {
12     for (int i=0; i<n; i++)
13       System.out.print("-");
14     System.out.println();
15   }
16
17   //.. garis model #3
18   static void buatGaris(char c, int n)
19   {
20     for (int i=0; i<n; i++)
21       System.out.print(c);
22     System.out.println();
23   }
24
25   //.. garis model #4
26   static void buatGaris(String s, int n)
```

```
27    {
28      for (int i=0; i<n; i++)
29        System.out.print(s);
30      System.out.println();
31    }
32
33    //.. Main program
34    public static void main(String[] args)
35    {
36      buatGaris();
37      buatGaris(30);
38      buatGaris('#', 20);
39      buatGaris("Java", 5);
40    }
41  }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
----------
------------------------------
####################
JavaJavaJavaJavaJava
```

6.6 Recursive Function

Fungsi rekursif adalah fungsi yang salah satu instruksinya adalah pemanggilan terhadap diri sendiri dengan mengirimkan data yang berbeda. Syarat yang harus dipenuhi adalah (1) ada titik henti atau kondisi tidak rekursif, dan (2) pada kondisi rekursif data yang diolah harus berbeda dibanding data induknya.

Persoalan yang sering dijadikan contoh ketika membahas fungsi rekursi adalah faktorial. Faktorial dari sebuah bilangan N adalah proses perkalian secara menurun dari Nx(N-1)x(N-2)x...x3x2x1.

3! = 3 x 2 x 1
2! = 2 x 1
1! = 1
0! = 1

Secara umum proses menghitung faktorial dapat dirumuskan sebagai berikut:

$$N! = \begin{cases} 1, \text{ untuk } N=0 \\ N \times (N-1)!, \text{ untuk } N>0 \end{cases}$$

Berdasarkan persamaan di atas, jika bilangan yang akan di-faktorial-kan lebih dari 0, proses menghitung faktorial ada di sebelah kiri dan kanan tanda sama dengan. Inilah yang disebut proses rekursif: penerima data dan data itu sendiri memiliki kesamaan proses.

Pada rumus di atas jika N = 0 proses akan langsung berhenti dengan mengembalikan nilai 1. Inilah yang disebut titik henti. Kondisi ini harus ada dalam fungsi yang rekursif agar proses di dalam fungsi tidak berputar-putar lepas kendali.

Perlu mendapat perhatian bahwa pada saat kondisi rekursif terjadi, komputer akan menggunakan *resource* untuk tujuan membuat fungsi baru yang memiliki struktur sama dengan fungsi pemanggil. Jadi kurang tepat jika dikatakan bahwa fungsi rekursif adalah fungsi yang memanggil dirinya sendiri; lebih tepat jika dikatakan bahwa fungsi rekursif adalah fungsi yang salah satu instruksinya adalah memanggil fungsi lain yang memiliki struktur sama dengan dirinya. Jika fungsi yang rekursif berukuran 100 byte, maka rekursi sekali akan menghabiskan 200 byte, rekursi kedua akan menghabiskan 300 byte, dan seterusnya, lama-lama memori akan habis dan program berhenti secara *abnormal*.

Program berikut ini merupakan implementasi pencarian nilai faktorial seperti telah dibahas di atas.

Program 6.15

```
1  public class Program615
2  {
3    //.. Fungsi faktorial
4    static int faktorial(int n)
5    {
6      if (n == 0)
7        return 1;
8      else
9        return n * faktorial(n-1);
10   }
11
12   //.. Main program
13   public static void main(String[] args)
14   {
15     int n = 5;
16     int f = faktorial(n);
17
18     System.out.println(n + "! = " + f);
19   }
20 }
```

Output program:

```
C:\WINDOWS\system32\cmd.exe
5! = 120
```

Instruksi pada baris 9 merupakan bagian dari fungsi faktorial(), akan tetapi di situ ada pemanggilan terhadap fungsi faktorial() itu sendiri. Inilah yang disebut rekursif. Secara teknis pemanggilan model ini akan mengalokasikan memori baru untuk fungsi yang dipanggil. Semakin banyak terjadi rekursi maka memori yang dihabiskan semakin banyak.

Program berikut ini digunakan untuk mencari bilangan pada posisi tertentu dari segitiga Pascal. Sebuah segitiga Pascal memiliki bentuk sebagai berikut:


```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

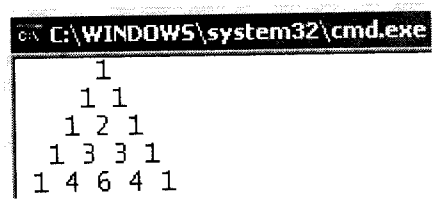
Segitiga tersebut adalah contoh jika banyaknya baris adalah 5. Angka di tepi kiri dan kanan pasti 1, dan angka selain itu diperoleh dari penjumlahan dua bilangan di atasnya. Dengan metode rekursi kita akan membuat fungsi yang akan mengembalikan nilai segitiga Pascal pada baris dan kolom tertentu.

Program 6.16

```
1  public class Program616
2  {
3    //.. Fungsi segitiga Pascal
4    static int pascal(int i, int j)
5    {
6      if ( (j == 0) || (i == j) )
7        return 1;
8      else
9        return pascal(i-1,j-1) + pascal(i-1,j);
10   }
11
12   //.. Main program
13   public static void main(String[] args)
14   {
15     int n = 5;
16
17     for (int i=0; i<n; i++)
18     {
19       for (int j=0; j<n-i; j++)
20         System.out.print(" ");
21
22       for (int j=0; j<=i; j++)
23         System.out.print(pascal(i,j) + " ");
24
25       System.out.println();
26     }
```

```
27      }
28  }
```

Output program:



Untuk nilai baris lebih dari 5 Anda harus memodifikasi instruksi baris 19 dan 20 yaitu upaya untuk menampilkan *space* sebelum angka segitiga agar tampilan segitiga berbentuk realistis.

6.7 Soal Latihan

1. Buatlah program dengan fungsi rekursif untuk menghitung KPK dan FPB!

2. Buatlah program dengan fungsi rekursif untuk menampilkan deret Fibonacci!

3. Buatlah program dengan fungsi rekursif untuk mengolah data bilangan sehingga hasilnya adalah bilangan yang sudah mendapat pemisah ribuan, misalnya data masukan 3200 akan menghasilkan output berupa 3,200!

4. Buatlah fungsi untuk mengolah data bilangan sehingga hasilnya adalah string berupa kalimat untuk bilangan tersebut, misalnya bilangan masukan adalah 2007 lalu hasil olahnya berupa kalimat "Dua ribu tujuh"!

DAFTAR PUSTAKA

Anuff, Ed. 1996. ***Java Sourcebook – Penuntun Pemrograman Java***. Terjemahan oleh Bambang Wisudawan. 1997. Yogyakarta : Penerbit Andi Offset.

Cornell, Gary dan Horstmann, Cay S. 1997. ***Core Java edisi Indonesia***. Terjemahan oleh Andreas Agus Setyabudi. 1997. Yogyakarta : Penerbit Andi Offset.

Deitel, H.M. dan Deitell, P.J. 1997. ***Java How to Program***. Prentice Hall, New Jersey.

Purnama, Rangsang. 2003. ***Tuntunan Pemrograman Java jilid 1***. Prestasi Pustaka, Jakarta.