

Aunur Rofiq Mulyanto, dkk.

# REKAYASA PERANGKAT LUNAK

untuk  
Sekolah Menengah Kejuruan

```
int _min(int a, int b, int c) {
    return min(min(a, b), c);
}
```

```
int **create_matrix(int Row, int Col) {
    int **array = new int*[Row];
    for(int i = 0; i < Row; ++i) {
        array[i] = new int[Col];
    }
    return array;
}
```

```
int **delete_matrix(int **array, int Row, int Col) {
    for(int i = 0; i < Row; ++i) {
        delete[] array[i];
    }
    return array;
}
```

Direktorat Pembinaan Sekolah Menengah Kejuruan

Direktorat Jenderal Manajemen Pendidikan Dasar dan Menengah

Departemen Pendidikan Nasional



```
for (*x, unsigned int m, const char *y,
```

Aunur R. Mulyanto

# REKAYASA PERANGKAT LUNAK JILID 2

**SMK**



**Direktorat Pembinaan Sekolah Menengah Kejuruan**

Direktorat Jenderal Manajemen Pendidikan Dasar dan Menengah  
Departemen Pendidikan Nasional

Hak Cipta pada Departemen Pendidikan Nasional  
Dilindungi Undang-undang

# REKAYASA PERANGKAT LUNAK JILID 2

Untuk SMK

Penulis : Aunur R. Mulyanto  
Perancang Kulit : Tim

Ukuran Buku : 17,6 x 25 cm

MUL MULYANTO,Aunur R.

Rekayasa Perangkat Lunak Jilid 1 untuk SMK /oleh Aunur R. Mulyanto ---- Jakarta : Direktorat Pembinaan Sekolah Menengah Kejuruan, Direktorat Jenderal Manajemen Pendidikan Dasar dan Menengah, Departemen Pendidikan Nasional, 2008.

v. 171 hlm

Daftar Pustaka : A1-A2

Glosarium : B1-B6

ISBN : 978-979-060-007-2

ISBN : 978-979-060-009-6

Diterbitkan oleh

**Direktorat Pembinaan Sekolah Menengah Kejuruan**

Direktorat Jenderal Manajemen Pendidikan Dasar dan Menengah

Departemen Pendidikan Nasional

Tahun 2008



## KATA SAMBUTAN

Puji syukur kami panjatkan kehadirat Allah SWT, berkat rahmat dan karunia Nya, Pemerintah, dalam hal ini, Direktorat Pembinaan Sekolah Menengah Kejuruan Direktorat Jenderal Manajemen Pendidikan Dasar dan Menengah Departemen Pendidikan Nasional, telah melaksanakan kegiatan penulisan buku kejuruan sebagai bentuk dari kegiatan pembelian hak cipta buku teks pelajaran kejuruan bagi siswa SMK. Karena buku-buku pelajaran kejuruan sangat sulit di dapatkan di pasaran.

Buku teks pelajaran ini telah melalui proses penilaian oleh Badan Standar Nasional Pendidikan sebagai buku teks pelajaran untuk SMK dan telah dinyatakan memenuhi syarat kelayakan untuk digunakan dalam proses pembelajaran melalui Peraturan Menteri Pendidikan Nasional Nomor 45 Tahun 2008 tanggal 15 Agustus 2008.

Kami menyampaikan penghargaan yang setinggi-tingginya kepada seluruh penulis yang telah berkenan mengalihkan hak cipta karyanya kepada Departemen Pendidikan Nasional untuk digunakan secara luas oleh para pendidik dan peserta didik SMK.

Buku teks pelajaran yang telah dialihkan hak ciptanya kepada Departemen Pendidikan Nasional ini, dapat diunduh (*download*), digandakan, dicetak, dialihmediakan, atau difotokopi oleh masyarakat. Namun untuk penggandaan yang bersifat komersial harga penjualannya harus memenuhi ketentuan yang ditetapkan oleh Pemerintah. Dengan ditayangkan *soft copy* ini diharapkan akan lebih memudahkan bagi masyarakat khususnya para pendidik dan peserta didik SMK di seluruh Indonesia maupun sekolah Indonesia yang berada di luar negeri untuk mengakses dan memanfaatkannya sebagai sumber belajar.

Kami berharap, semua pihak dapat mendukung kebijakan ini. Kepada para peserta didik kami ucapkan selamat belajar dan semoga dapat memanfaatkan buku ini sebaik-baiknya. Kami menyadari bahwa buku ini masih perlu ditingkatkan mutunya. Oleh karena itu, saran dan kritik sangat kami harapkan.

Jakarta, 17 Agustus 2008  
Direktur Pembinaan SMK



## PENGANTAR PENULIS

Dengan segala kerendahan hati, kami mengucapkan syukur kepada Allah SWT. Karena hanya dengan lindungan, rahmat dan karuniaNya-lah maka buku ini dapat diselesaikan.

Buku yang berjudul 'Rekayasa Perangkat Lunak' merupakan buku yang disusun untuk memenuhi kebutuhan buku pegangan bagi siswa Sekolah Menengah Kejuruan. Khususnya pada program keahlian Rekayasa Perangkat Lunak. Buku ini memuat uraian yang mengacu pada standar kompetensi dan kompetensi dasar Rekayasa Perangkat Lunak untuk siswa SMK mulai dari kelas X, XI sampai dengan kelas XII.

Tiap bab berisi teori yang harus dipahami secara benar oleh peserta didik dan disertai dengan contoh-contoh soal yang relevan dengan teori tersebut. Selain itu terdapat juga soal-soal yang didasarkan pada konsep dan teori yang dibahas sebagai alat uji untuk mengukur kemampuan peserta didik dalam penguasaan materi tersebut.

Dalam mengembangkan buku ini, penulis berupaya agar materi yang disajikan sesuai dengan kebutuhan kompetensi yang harus dicapai. Oleh karenanya, selain dari hasil pemikiran dan pengalaman penulis sebagai pengajar dan praktisi Rekayasa Perangkat Lunak, materi yang dikembangkan juga diperkaya dengan referensi-referensi lain yang sesuai.

Pada kesempatan ini penulis ingin menyampaikan rasa terima kasih kepada semua pihak yang mendukung buku ini dapat diterbitkan. Mudah-mudahan buku ini dapat bermanfaat bagi peserta didik dalam mengembangkan kemampuannya. Penulis menyadari bahwa buku ini masih perlu dikembangkan terus menerus, sehingga saran dari berbagai pihak pengguna buku ini sangat diharapkan.

Penulis





---

---

## DAFTAR ISI

---

---

KATA SAMBUTAN .....	i
PENGANTAR PENULIS .....	ii
DAFTAR ISI .....	iii
PETUNJUK PENGGUNAAN BUKU .....	vi
BAB 1 PENDAHULUAN .....	1
1.1. PENGERTIAN REKAYASA PERANGKAT LUNAK.....	2
1.2. TUJUAN REKAYASA PERANGKAT LUNAK.....	2
1.3. RUANG LINGKUP .....	3
1.4. REKAYASA PERANGKAT LUNAK DAN DISIPLIN ILMU KOMPUTER .....	4
1.5. REKAYASA PERANGKAT LUNAK DAN DISIPLIN ILMU LAIN..	8
1.6. PERKEMBANGAN REKAYASA PERANGKAT LUNAK.....	8
1.7. PROFESI DAN SERTIFIKASI .....	9
1.8. REKAYASA PERANGKAT LUNAK DAN PEMECAHAN MASALAH.....	10
1.9. RINGKASAN.....	14
1.10. SOAL-SOAL LATIHAN.....	15
BAB 2 METODE REKAYASA PERANGKAT LUNAK.....	17
2.1. MODEL PROSES REKAYASA PERANGKAT LUNAK .....	17
2.2. TAHAPAN REKAYASA PERANGKAT LUNAK .....	24
2.3. RINGKASAN.....	31
2.4. SOAL-SOAL LATIHAN.....	32
BAB 3 ELEKTRONIKA DAN SISTEM KOMPUTER .....	33
3.1. DASAR ELEKTRONIKA.....	34
3.2. ELEKTRONIKA DIGITAL .....	36
3.3. SISTEM KOMPUTER.....	39
3.4. RINGKASAN.....	50
3.5. SOAL-SOAL LATIHAN.....	51
BAB 4 SISTEM OPERASI .....	53
4.1. PENGERTIAN SISTEM OPERASI .....	54
4.2. JENIS-JENIS SISTEM OPERASI .....	60
4.3. MENYIAPKAN DAN MENJALANKAN SISTEM OPERASI .....	68
4.4. BEKERJA DALAM KOMPUTER JARINGAN .....	86
4.5. RINGKASAN.....	92
4.6. SOAL-SOAL LATIHAN.....	92
BAB 5 ALGORITMA PEMROGRAMAN DASAR .....	93
5.1. VARIABEL, KONSTANTA DAN TIPE DATA .....	94
5.2. STRUKTUR ALGORITMA PEMROGRAMAN .....	101
5.3. PENGELOLAAN ARRAY .....	121
5.4. OPERASI FILE .....	126

5.5.	RINGKASAN.....	128
5.6.	SOAL-SOAL LATIHAN.....	129
BAB 6 ALGORITMA PEMROGRAMAN LANJUTAN.....		131
6.1.	ARRAY MULTIDIMENSI .....	132
6.2.	PROSEDUR DAN FUNGSI.....	136
6.3.	RINGKASAN.....	138
6.4.	SOAL-SOAL LATIHAN.....	139
BAB 7 PEMROGRAMAN APLIKASI DENGAN VB & VB.NET.....		141
7.1.	DASAR-DASAR VISUAL BASIC.....	142
7.2.	AKSES DAN MANIPULASI BASIS DATA DENGAN VISUAL BASIC .....	163
7.3.	TEKNOLOGI COM .....	166
BAB 8 PEMROGRAMAN BERORIENTASI OBYEK DENGAN JAVA ...		169
8.1.	KONSEP PEMROGRAMAN BERORIENTASI OBYEK .....	170
8.2.	Pengenalan pada Java.....	172
8.3.	TIPE DATA, VARIABEL, DAN PERNYATAAN INPUT/OUTPUT (I/O) .....	176
8.4.	OPERATOR.....	179
8.5.	STRUKTUR KONTROL PROGRAM .....	182
8.6.	EXCEPTION HANDLING .....	186
8.7.	MULTI-THREADING .....	191
8.8.	APLIKASI PEMROGRAMAN BERORIENTASI OBYEK DENGAN JAVA .....	194
BAB 9 PEMROGRAMAN APLIKASI DENGAN C++ .....		217
9.1.	DASAR-DASAR PEMROGRAMAN C++ .....	218
9.2.	FUNGSI DALAM C++.....	230
9.3.	POINTER DAN ARRAY .....	233
9.4.	KELAS .....	240
9.5.	MERANCANG APLIKASI BERORIENTASI OBYEK .....	248
BAB 10 DASAR-DASAR SISTEM BASIS DATA .....		253
10.1.	DATA, BASIS DATA DAN SISTEM MANAJEMEN BASIS DATA .....	254
10.2.	ENTITY-RELATIONSHIP DIAGRAM.....	262
10.3.	BASIS DATA RELASIONAL .....	268
10.4.	RINGKASAN.....	277
10.5.	SOAL-SOAL LATIHAN.....	278
BAB 11 APLIKASI BASIS DATA BERBASIS MICROSOFT ACCESS ..		279
11.1.	MENU-MENU UMUM APLIKASI BASIS DATA .....	280
11.2.	TABEL.....	285
11.3.	QUERY .....	290
11.4.	FORM .....	301
11.5.	REPORT .....	312
11.6.	RINGKASAN.....	320

11.7. SOAL-SOAL LATIHAN.....	321
BAB 12 BASIS DATA BERBASIS SQL .....	323
12.1. SEKILAS TENTANG SQL .....	324
12.2. MEMPERSIAPKAN PERANGKAT LUNAK BERBASIS SQL.....	325
12.3. MENU / FITUR UTAMA.....	328
12.4. PEMBUATAN DAN PENGISIAN TABEL .....	329
12.5. OPERASI PADA TABEL DAN VIEW .....	332
12.6. MENGGUNAKAN T-SQL .....	339
12.7. FUNGSI, PROCEDURE DAN TRIGGER .....	349
12.9. RINGKASAN.....	357
12.10. SOAL-SOAL LATIHAN.....	358
BAB 13 DESAIN WEB STATIS DAN HTML .....	359
13.1. KONSEP DASAR DAN TEKNOLOGI WEB.....	360
13.2. PERSIAPAN PEMBUATAN <i>WEB</i> .....	362
13.3. MEMBUAT DAN MENGUJI HALAMAN <i>WEB</i> .....	367
13.4. HTML .....	369
13.5. RINGKASAN.....	387
13.6. SOAL-SOAL LATIHAN.....	387
BAB 14 DINAMIS BERBASIS JSP .....	389
14.1 DASAR WEB DINAMIS.....	390
14.2 RINGKASAN.....	413
14.3 SOAL-SOAL LATIHAN.....	414

LAMPIRAN A DAFTAR PUSTAKA

LAMPIRAN B GLOSARIUM, DAFTAR WEBSITE

# PETUNJUK PENGGUNAAN BUKU

## A. Deskripsi Umum

Buku ini diberi judul “Rekayasa Perangkat Lunak”, sama dengan salah satu program keahlian pada Sekolah Menengah Kejuruan (SMK). Meskipun demikian, sebenarnya isi dari buku ini tidak secara khusus membahas tentang Rekayasa Perangkat Lunak. Dari sisi pandang bidang Ilmu Komputer ada lima sub-bidang yang tercakup dalam buku ini, yaitu sub-bidang Rekayasa Perangkat Lunak, Sistem Operasi, Algoritma dan Struktur Data, Bahasa Pemrograman dan Basis Data. Hal ini disesuaikan dengan kurikulum tingkat SMK untuk Program Keahlian Rekayasa Perangkat Lunak.

Pokok bahasan tentang Rekayasa Perangkat Lunak secara umum membahas dasar-dasar pengertian Rekayasa Perangkat Lunak, masalah dan pemecahan masalah, dan metode-metode pengembangan perangkat lunak. Pembahasan tentang sub-bidang Sistem Operasi berisi sistem computer, sistem operasi dan bekerja dalam jaringan computer. Cakupan materi algoritma meliputi algoritma dasar dan algoritma lanjutan. Sub bidang Bahasa Pemrograman mengambil porsi yang cukup besar, meliputi pemrograman GUI dengan VB & VB.Net, pemrograman Java, pemrograman C++, pemrograman berorientasi obyek dan Pemrograman berbasis web. Sub-bidang terakhir yang menjadi bagian dari buku ini adalah Basis Data dengan cakupan tentang system basis data, pemodelan konseptual, basis data relasional, Microsoft Access dan SQL.

## B. Peta Kompetensi

Secara umum, buku ini mengacu pada Standar Kompetensi dan Kompetensi Dasar (SKKD) bagi SMK seperti berikut.

1. Menggunakan algoritma pemrograman tingkat dasar
2. Menggunakan algoritma pemrograman tingkat lanjut
3. Mengoperasikan aplikasi basis data
4. Membuat aplikasi berbasis Microsoft Access
5. Menguasai teknik elektronika dasar
6. Menguasai teknik elektronika digital
7. Membuat file dengan HTML sesuai spesifikasi
8. Menerapkan dasar-dasar pembuatan web statis tingkat dasar
9. Membuat program aplikasi menggunakan VB dan VB.NET
10. Membuat paket software aplikasi
11. Melakukan pemrograman data deskripsi (SQL – Structured Query Language) tingkat dasar
12. Mengoperasikan bahasa pemrograman data deskripsi (SQL) tingkat lanjut
13. Membuat halaman web dinamis tingkat dasar
14. Membuat halaman web dinamis tingkat lanjut
15. Membuat program aplikasi web menggunakan JSP

16. Membuat program aplikasi basis data menggunakan XML
17. Membuat program basis data menggunakan Microsoft (SQL Server)
18. Membuat program basis data menggunakan PL/SQL (Oracle)
19. Membuat program aplikasi menggunakan C++
20. Menjelaskan sistem peripheral
21. Membuat program dalam bahasa pemrograman berorientasi obyek
22. Membuat program aplikasi menggunakan Java
23. Mengoperasikan sistem operasi komputer berbasis teks dan GUI

Dalam penyajian buku ini, bab-bab tidak disusun berdasarkan SKKD, akan tetapi disusun berdasarkan urutan materi pokok bahasan. Sehingga di beberapa bab berisi gabungan dari beberapa standar kompetensi. Atau satu kompetensi dasar mungkin berada tidak pada kelompok standar kompetensi seperti pada daftar SKKD, tetapi berada pada sub bab yang lain.

Kesesuaian SKKD dan isi bab dapat dilihat pada table berikut ini.

Kode Kompetensi	Kompetensi	Bab Terkait
<i>ELKA-MR.UM.001.A</i>	Menguasai Teknik Dasar Elektronika	<b>3</b>
<i>ELKA.MR.UM.004.A</i>	Menguasai Dasar Elektronika Digital dan Komputer	<b>3</b>
<i>TIK.PR02.001.01</i>	Menggunakan algoritma pemograman tingkat dasar	<b>5</b>
<i>TIK.PR02.002.01</i>	Menggunakan algoritma pemograman tingkat lanjut	<b>6</b>
<i>HDW.OPR.103.(1).A</i>	Mengoperasikan sistem operasi jaringan komputer berbasis teks	<b>4</b>
<i>HDW.OPR.104.(1).A</i>	Mengoperasikan sistem operasi jaringan komputer berbasis GUI	<b>4</b>
<i>TIK.PR02.020.01</i>	Mengoperasikan aplikasi basis Data	<b>10 dan 11</b>
<i>TIK.PR08.004.01</i>	Membuat aplikasi Berbasis Microsoft Acces	<b>11</b>
<i>TIK.PR08.024.01</i>	Membuat dokumen dengan HTML sesuai spesifikasi	<b>13</b>
<i>TIK.PR08.027.01</i>	Menerapkan dasar-dasar pembuatan web statis tingkat dasar.	<b>13</b>
<i>TIK.PR08.003.01</i>	Membuat program aplikasi menggunakan VB & VB.NET	<b>7</b>
<i>TIK.PR02.016.01</i>	Membuat paket software Aplikasi	<b>7</b>
<i>TIK.PR03.001.01</i>	Mengoperasikan bahasa pemrograman data deskripsi (SQL) tingkat dasar	<b>12</b>
<i>TIK.PR03.002.01</i>	Mengoperasikan bahasa pemrograman data deskripsi (SQL) tingkat Lanjut	<b>12</b>
<i>TIK.PR04.002.01</i>	Membuat Halaman Web dinamis tingkat dasar	<b>13</b>
<i>TIK.PR04.003.01</i>	Membuat Halaman Web dinamis tingkat Lanjut.	<b>13</b>

Kode Kompetensi	Kompetensi	Bab Terkait
<i>TIK.PR02.009.01</i>	Mengoperasikan bahasa pemrograman berorientasi obyek	<b>8</b>
<i>TIK.PR08.012.01</i>	Membuat program aplikasi menggunakan Java	<b>8</b>
<i>TIK.PR08.001.01</i>	Membuat program aplikasi menggunakan C++	<b>9</b>
<i>TIK.PR06.003.01</i>	Menjelaskan sistem Peripherals	<b>3</b>
<i>TIK.PR08.005.01</i>	Membuat program basis data menggunakan PL/SQL	<b>10 dan 12</b>
<i>TIK.PR08.006.01</i>	Membuat program basis data menggunakan SQL Server	<b>12</b>
<i>TIK.PR08.008.01</i>	Membuat program aplikasi web berbasis JSP	<b>14</b>

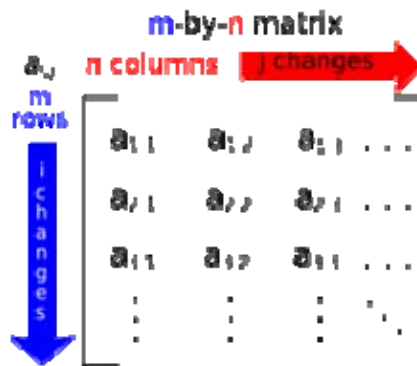
### C. Cara Menggunakan Buku

Buku ini secara khusus ditujukan kepada siswa dan guru SMK untuk program keahlian RPL. Namun demikian, buku ini juga terbuka bagi pembaca umum yang berminat dalam dunia RPL, Algoritma dan Pemrograman, Basis Data dan Internet. Bagi siswa, buku ini dapat dijadikan buku pegangan, karena ini buku ini menyediakan bahan-bahan pelajaran yang cukup lengkap untuk mata pelajaran selama tiga tahun di bangku sekolah. Beberapa bagian dari buku ini mungkin memerlukan buku-buku bantu lainnya untuk lebih memperkaya wawasan dan peningkatan kemampuan. Sedangkan bagi guru, buku ini dapat digunakan sebagai buku referensi untuk menyusun modul-modul ajar bagi anak didiknya.

Buku ini disusun sedemikian rupa agar siswa dapat belajar secara mandiri dan terdorong untuk mencoba secara langsung. Oleh karena itu dalam buku ini, akan banyak dijumpai ilustrasi baik yang berupa gambar, skema maupun *listing program*. Hal ini dimaksudkan agar siswa dapat dengan mudah memahami penjelasan ataupun penerapan suatu konsep tertentu. Bahkan pada bagian akhir bab diakhiri dengan soal-soal latihan dari pokok bahasan pada bab tersebut.

## BAB 6 ALGORITMA PEMROGRAMAN LANJUTAN

Gambar 6.1. di samping ini saya yakin kalian pernah melihatnya. Ya, inilah notasi umum dari salah satu teori matematika, yaitu matrik. Operasi matrik, merupakan operasi yang banyak digunakan dalam banyak aktifitas yang membutuhkan perhitungan dengan model banyak bilangan. Cobalah buka kembali buku matematika kalian yang membahas matrik kalian akan melihat banyak sekali yang bisa kita lakukan dengan matrik.



Gambar 6.1. Notasi matrik.

Pernahkan kalian mendengar perangkat lunak bernama MATLAB? Perangkat lunak yang didasarkan pada operasi matrik ini sangat populer dikalangan ilmuwan, insinyur dan orang-orang yang banyak berkecimpung dengan perhitungan. Dengan menggunakan array kalian juga bisa membuat program yang dapat melakukan operasi matrik seperti halnya MATLAB.

Standar kompetensi algoritma pemrograman lanjutan terdiri atas empat kompetensi dasar yaitu, menggunakan array multidimensi dan menggunakan prosedur dan fungsi. Dalam penyajian pada buku ini, setiap kompetensi dasar memuat uraian materi, dan latihan. Ringkasan diletakkan pada setiap akhir bab. Sebelum mempelajari kompetensi ini ingatlah kembali prinsip pemecahan masalah, algoritma pemrograman dasar dan materi-materi pendukung dari mata pelajaran matematika.

Pada akhir bab, tercantum soal-soal latihan yang disusun dari soal-soal yang mudah hingga soal-soal yang sulit. Latihan soal ini digunakan untuk mengukur kemampuan terhadap kompetensi dasar ini. Artinya setelah mempelajari kompetensi dasar ini secara mandiri dengan bimbingan guru sebagai fasilitator, ukurlah sendiri kemampuan dengan mengerjakan soal-soal latihan tersebut.

## TUJUAN

Setelah mempelajari bab ini diharapkan kalian akan mampu :

- o Menggunakan array multidimensi
- o Menggunakan prosedur dan fungsi

## 6.1. ARRAY MULTIDIMENSI

### 6.1.1. Pengertian Array Multidimensi

Array satu dimensi seperti pada Bab 5 sangat baik untuk menyimpan data sejenis yang berurutan, namun bagaimana bila kita ingin menyimpan daftar kota dengan temperature rata-ratanya secara bersama-sama, atau menyimpan data nama Siswa dengan nilai ujiannya? Pada kasus seperti ini kita dapat menggunakan dua array satu dimensi, satu array untuk menyimpan nama dan satu array untuk menyimpan nilai. Namun ini bukanlah pilihan yang baik karena akan menyulitkan dan membuat kode program menjadi tidak efisien. Pilihan yang lebih baik adalah dengan menggunakan pendekatan Array Multidimensi. Kita dapat menyimpan dengan menggunakan array dua dimensi untuk kasus di atas. Perhatikan gambar berikut ini untuk melihat perbedaan ***dua array satu dimensi*** dengan **array dua dimensi**.

Nama(4)	Nilai(4)		NilaiSiswa (4,1)	
Joni	70	0	Joni	70
Rudi	80	1	Rudi	80
Sari	45	2	Sari	45
Dono	56	3	Dono	56
Indro	77	4	Indro	77

Dua array satu dimensi

Array dua dimensi

Gambar 6.2. Perbedaan array satu dimensi dan dua dimensi

Array dua dimensi mempunyai dua indeks. Indeks yang pertama menunjukkan baris sedangkan indeks yang kedua adalah kolom. Pada Gambar 6.2, variable array NilaiSiswa memiliki dua indeks yaitu indeks pertama 4 yang menyatakan nilai indeks maksimal untuk baris adalah 4 (atau ada 5 baris karena indeks baris pertama bernilai 0), sedangkan indeks kedua adalah 1 yang menunjukkan nilai indeks maksimal untuk kolom adalah 1 (atau ada 2 kolom karena indeks kolom pertama bernilai 0). Untuk mendeklarasikan array dua dimensi dapat digunakan cara sebagai berikut :

**Dim NilaiSiswa(4,1)**

Sedangkan untuk mengakses nilai pada array dua dimensi dapat digunakan seperti contoh berikut.

Contoh 6.1. Membaca nilai array multidimensi



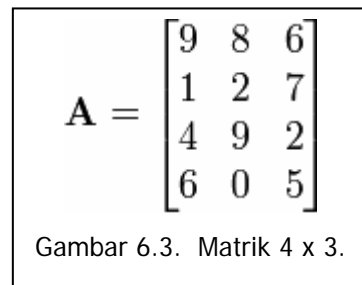
```
Print NilaiSiswa(3,0)
Print NilaiSiswa(3,1)
```

Pernyataan pertama dari contoh 6.1, akan menghasilkan output “Dono” (Baris ke 4 dan kolom pertama) sedangkan pernyataan kedua menghasilkan output 56 (Baris ke 4 kolom kedua).

Keuntungan menggunakan array multidimensi adalah secara konseptual, array ini lebih mudah dikelola. Sebagai contoh kita ingin membuat program permainan dan kita ingin mencari posisi dari suatu tempat pada sebuah papan permainan. Setiap bujursangkar dapat diidentifikasi dengan menggunakan dua angka, yaitu koordinat horizontal dan vertikalnya (atau baris dan kolomnya). Struktur seperti ini adalah tipikal penggunaan array dua dimensi. Koordinat horizontal adalah indeks barisnya sedangkan koordinat vertical adalah indeks kolomnya. Bentuk array multidimensi ini dapat dikembangkan menjadi lebih dari dua dimensi. Pernyataan **Dim Matrix(9,9,9)** akan membuat array multidimensi yang memiliki 1000 elemen (10x10x10).

### 6.1.2. Operasi Matriks dengan Array Multidimensi

Seperti disinggung di awal bab ini, kita bisa menggunakan array untuk melakukan operasi matrik. Perhatikan Gambar 6.3 berikut ini.


$$A = \begin{bmatrix} 9 & 8 & 6 \\ 1 & 2 & 7 \\ 4 & 9 & 2 \\ 6 & 0 & 5 \end{bmatrix}$$

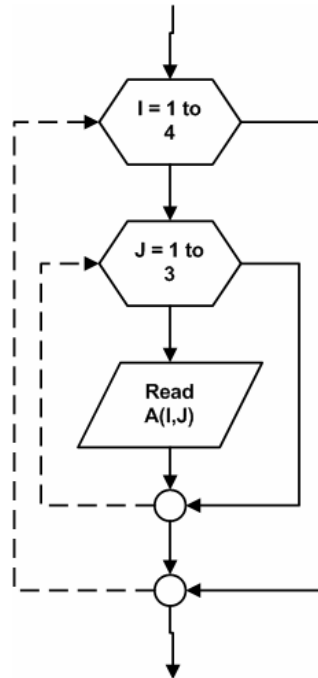
Gambar 6.3. Matrik 4 x 3.

Gambar 6.3 ini menunjukkan matrik dua dimensi yang terdiri dari 4 baris dan 3 kolom. Atau biasa dinotasikan sebagai  $A_{4 \times 3}$ . Hal ini identik dengan array multidimensi dengan yang kita definisikan sebagai  $A(3, 2)$ . Perhatikan kembali angka pada indeks array selalu lebih kecil satu dari jumlah sesungguhnya karena indeks selalu dimulai dengan 0. Sedangkan indeks pada matrik dimulai pada angka 1. Untuk

membuat array atau matrik seperti pada Gambar 6.3, maka kita memerlukan struktur pengulangan. Bentuk for dapat kita gunakan karena kita telah tahu dengan pasti berapa baris dan berapa kolomnya. Gambar berikut menyajikan flowchart untuk membuat matrik pada Gambar 6.3.

Pada Gambar 6.4. kita melihat dua buah variabel counter yaitu I dan J. I digunakan untuk membuat indeks pada baris yaitu 4 baris (1 to 4), sedangkan J digunakan untuk membuat indeks pada kolom yaitu 3 kolom (1 to 3). Pembacaan data akan berlangsung sebagai berikut:

```
A(1,1) = ....
A(1,2) = ....
A(1,3) = ....
A(2,1) = ....
A(2,2) = ....
A(2,3) = .... dan seterusnya.
```



Gambar 6.4. Algoritma untuk membuat matrik 4 x 3.

#### Contoh 6.2. Operasi penjumlahan pada matrik.

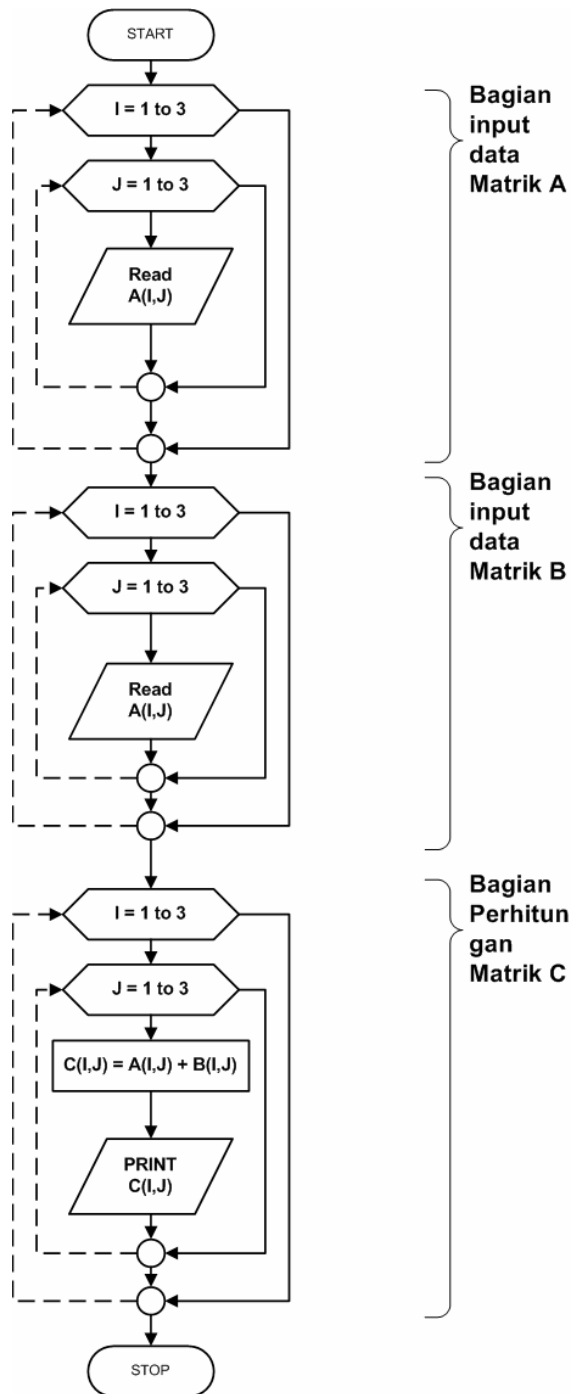
Perhatikan matrik berikut ini:

$$\begin{bmatrix} 1 & 3 & 1 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 1+5 \\ 1+7 & 0+5 & 0+0 \\ 1+2 & 2+1 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 6 \\ 8 & 5 & 0 \\ 3 & 3 & 3 \end{bmatrix}.$$

Operasi matriks di atas adalah operasi penjumlahan dua buah matriks dengan dimensi yang sama yaitu 3x3. Bagaimanakah algoritma penyelesaiannya?

*Penyelesaian:*

Pada contoh soal di atas ada dua buah matriks misalkan A matriks pertama dan B adalah matriks kedua. Selain itu juga dibutuhkan matriks ke tiga yaitu C sebagai hasil penjumlahan matriks A dan B. Gambar 6.5. menunjukkan algoritma penjumlahan dua buah matriks.



Gambar 6.5. Algoritma penjumlahan dua buah matrik

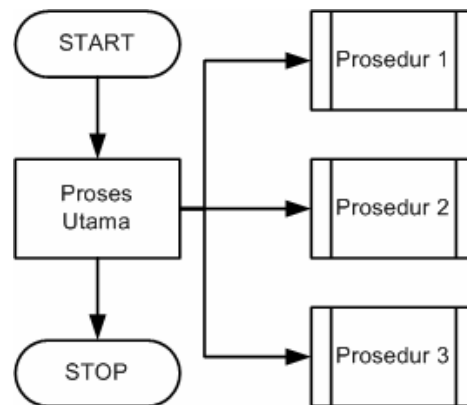
## 6.2. PROSEDUR DAN FUNGSI

Setiap bahasa pemrograman selalu menyediakan fungsi-fungsi yang sudah didefinisikan oleh bahasa pemrograman tersebut (*built-in function*). Namun ada kalanya kita memerlukan suatu prosedur tertentu yang kita gunakan berulang kali dan tidak tersedia dalam built-in function.

**Prosedur** adalah sekumpulan perintah yang merupakan bagian dari program yang lebih besar yang berfungsi mengerjakan suatu tugas tertentu. Prosedur atau kadang disebut subrutin / subprogram biasanya relative independent terhadap bagian kode program yang lain. Atau sebenarnya prosedur dapat berdiri sendiri. Keuntungan menggunakan prosedur adalah :

- o mengurangi duplikasi kode program.
- o memberikan kemungkinan penggunaan kembali kode untuk program yang lain.
- o memecah masalah yang rumit dalam masalah-masalah yang lebih kecil dan lebih mudah diselesaikan (lihat prinsip problem reduction pada Bab 2).
- o membuat kode program lebih mudah dibaca.
- o dapat digunakan untuk menyembunyikan detail program.

Pada flowchart untuk menuliskan prosedur digunakan notasi Predefined Process (lihat Bab 5 untuk notasi flowchart). Secara skematis penggunaan prosedur dapat dilihat pada Gambar 6.6.



Gambar 6.6. Skema penggunaan prosedur.

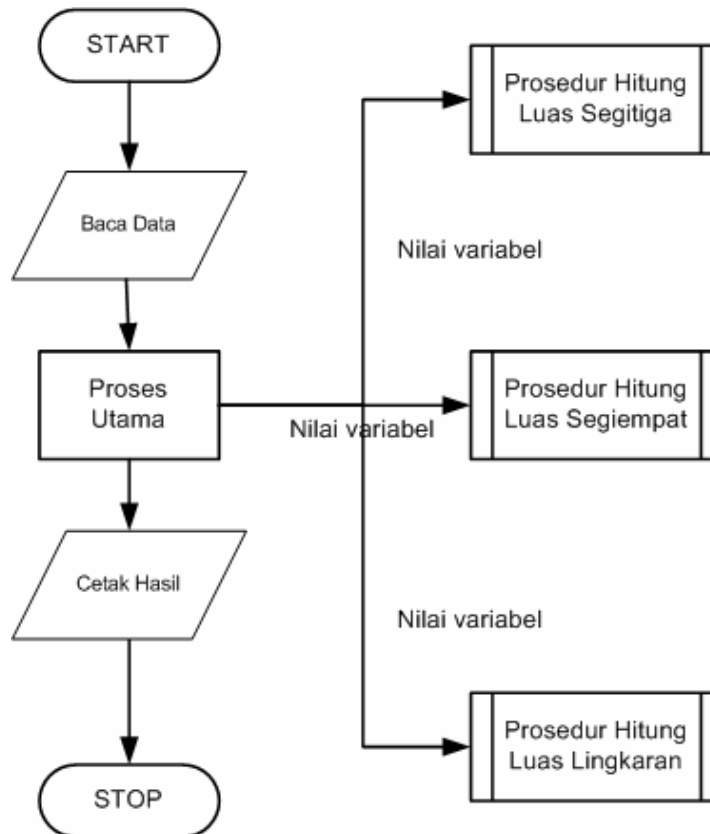
Gambar 6.6. menunjukkan ada proses utama yang terjadi dan ada prosedur yang sebenarnya merupakan bagian dari proses utama ini. Ketika proses utama membutuhkan suatu tugas tertentu maka proses utama akan memanggil prosedur tertentu menyelesaikan tugas tersebut. Perhatikan contoh berikut:

### Contoh 6.3. Prosedur

Buatlah algoritma menghitung luas segitiga, segiempat, dan lingkaran.

Penyelesaian:

Untuk membuat algoritma ini kita dapat memandang proses perhitungan luas segitiga, luas segiempat dan luas lingkaran sebagai bagian program yang berdiri sendiri. Kita dapat membuat prosedur untuk masing-masing proses. Dan kita akan memanggil prosedur tersebut dari proses utama (Gambar 6.7).



Gambar 6.7. Penyelesaian contoh 6.3.

Urutan proses pada Gambar 6.7 adalah sebagai berikut.

- o Pembacaan data
- o Pada proses utama akan terjadi pengecekan pada data yang dibaca,
- o Apabila data yang dibaca adalah untuk segitiga, maka proses utama akan memanggil prosedur hitung luas segitiga dengan

membawa nilai variable yang diperlukan oleh prosedur luas hitung segitga.

- o Proses perhitungan luas segitiga hanya dilakukan pada prosedur tersebut.
- o Setelah proses perhitungan maka hasil perhitungan akan dibawa kembali ke proses utama untuk dicetak hasilnya.

Urutan proses yang sama juga terjadi jika data yang dibaca adalah untuk segiempat atau lingkaran.

Prosedur yang baik memiliki ciri-ciri sebagai berikut :

- o Hanya memiliki satu fungsi tujuan (*logical inherent*).

Sebuah prosedur sebaiknya hanya memiliki satu fungsi tujuan dan tidak bercampur dengan tujuan-tujuan lain. Hal ini untuk membuat prosedur lebih focus sehingga tujuan akan dapat tercapai dengan baik.

- o Tidak tergantung pada prosedur lain (*independent*).

Sebuah prosedur harusnya bersifat mandiri, artinya prosedur ini dapat berjalan dan diuji tanpa menunggu bagian lainnya selesai. Selain itu variable-variabel yang digunakan dalam prosedur tersebut tidak mempengaruhi variable-variabel yang digunakan pada bagian lain di keseluruhan program.

- o Berukuran kecil (*small size*).

Yang dimaksud ukuran disini adalah panjang algoritma atau panjang kode program pada suatu prosedur. Ukuran kecil akan mudah dibaca dan diperbaiki. Apabila sebuah modul sudah terlalu besar maka sebaiknya dipertimbangkan untuk dipecah-pecah menjadi beberapa modul yang lebih kecil.

### 6.3. Ringkasan

- Array dua dimensi mempunyai dua indeks. Indeks yang pertama menunjukkan baris sedangkan indeks yang kedua adalah kolom.
- Pembacaan data dan penulisan data pada array multidimensi dilakukan dengan langsung menunjuk pada nomor indeksnya
- **Prosedur** adalah sekumpulan perintah yang merupakan bagian dari program yang lebih besar yang berfungsi mengerjakan suatu tugas tertentu.

#### **6.4. Soal-Soal Latihan**

1. Perhatikan contoh 6.1 di atas. Buatlah algoritma untuk operasi pengurangan dua buah matriks.
2. Buatlah juga algoritma untuk operasi perkalian dua buah matriks.
3. Dengan menggunakan prosedur buatlah sebuah algoritma yang membaca data matrik kemudian memilih melakukan operasi penjumlahan, pengurangan dan perkalian dua buah matrik.





## BAB 7 PEMOGRAMAN APLIKASI DENGAN VB & VB.NET



Gambar 7.1. Aplikasi yang dibangun dengan Visual Basic.

Perhatikan Gambar 7.1. di atas. Tampilan pada gambar tersebut menunjukkan sebuah aplikasi yang menampilkan sekumpulan data pada tabel. Suatu ketika kalian akan menjumpai aplikasi yang serupa. Aplikasi di atas dapat dibangun dengan menggunakan Visual Basic secara cepat, tanpa harus dipusingkan dengan masalah pembuatan tabel, menu, tombol atau yang lainnya. Tinggal klik, seret, letakkan, atur posisi kemudian buat kodenya, aplikasi sudah langsung bisa dijalankan.

Standar kompetensi pemrograman dengan VB dan VB.Net dasar terdiri atas tiga kompetensi dasar. Dalam penyajian pada buku ini, setiap kompetensi dasar memuat uraian materi, dan latihan. Ringkasan diletakkan pada setiap akhir bab kemudian dilanjutkan dengan soal-soal latihan. Kompetensi dasar pada bab ini adalah menjelaskan dasar-dasar Visual Basic, mengakses dan memanipulasi data dengan Visual Basic, dan menerapkan teknologi COM. Sebelum mempelajari kompetensi ini ingatlah kembali sistem operasi, prinsip pemecahan masalah, algoritma pemrograman dasar dan lanjutan dan materi-materi pendukung dari mata pelajaran matematika.

## TUJUAN

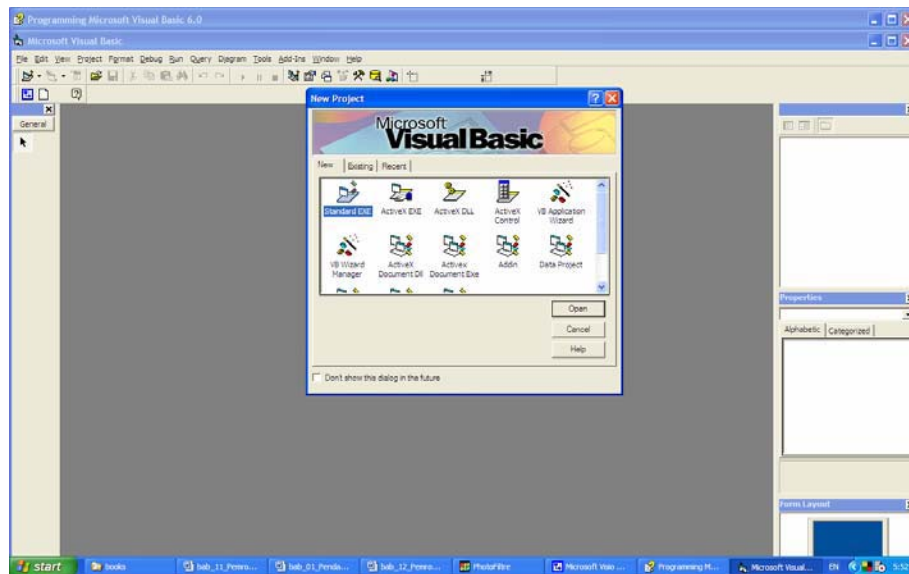
Setelah mempelajari bab ini diharapkan pembaca akan mampu :

- o Menjelaskan dasar-dasar Visual Basic
- o Mengakses dan memanipulasi data dengan Visual Basic
- o Menerapkan teknologi COM

### 7.1. Dasar-dasar Visual Basic

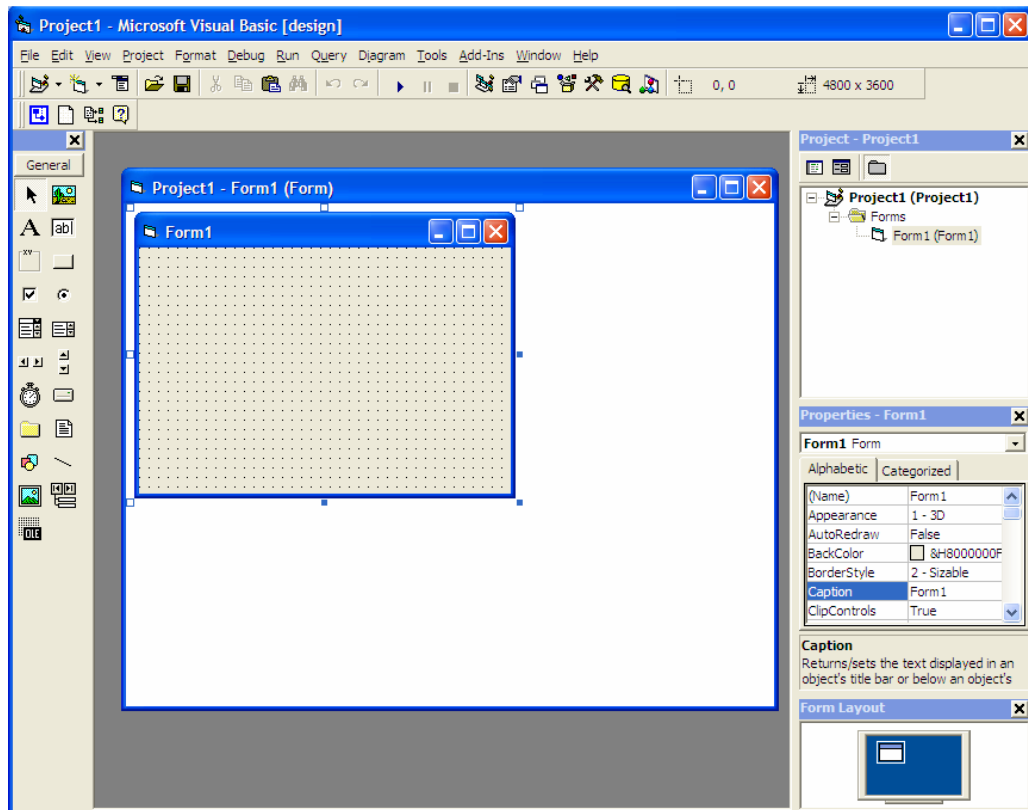
Visual Basic (VB) adalah salah satu bahasa pemrograman komputer. Bahasa pemrograman VB, yang dikembangkan oleh Microsoft sejak tahun 1991, merupakan pengembangan dari pendahulunya yaitu bahasa pemrograman BASIC (*Beginner's All-purpose Symbolic Instruction Code*) yang dikembangkan pada era 1950-an. VB adalah salah satu *development tools* untuk membangun aplikasi dalam lingkungan Windows. Dalam pengembangan aplikasi, Visual Basic menggunakan pendekatan Visual untuk merancang *user interface* dalam bentuk *form*, sedangkan untuk *codingnya* menggunakan dialek bahasa Basic yang cenderung mudah dipelajari. Visual Basic telah menjadi *tools* yang terkenal bagi para pemula maupun para *developer*. Namun ada kelemahan dari VB yaitu pada kinerja yang relative terasa lebih lambat dibanding dengan bahasa pemrograman lain. Namun dengan perkembangan *processor* dan *main memory* yang semakin cepat permasalahan ini menjadi tidak begitu penting.

Visual Basic adalah perangkat lunak yang berjalan di atas *platform* sistem operasi Microsoft Windows. Untuk memulai Visual Basic dapat dilakukan dengan mengklik **Start -> Programs -> Microsoft Visual Studio 6 -> Microsoft Visual Basic**. Tampilan awal Visual Basic akan tampak seperti Gambar 7.2.



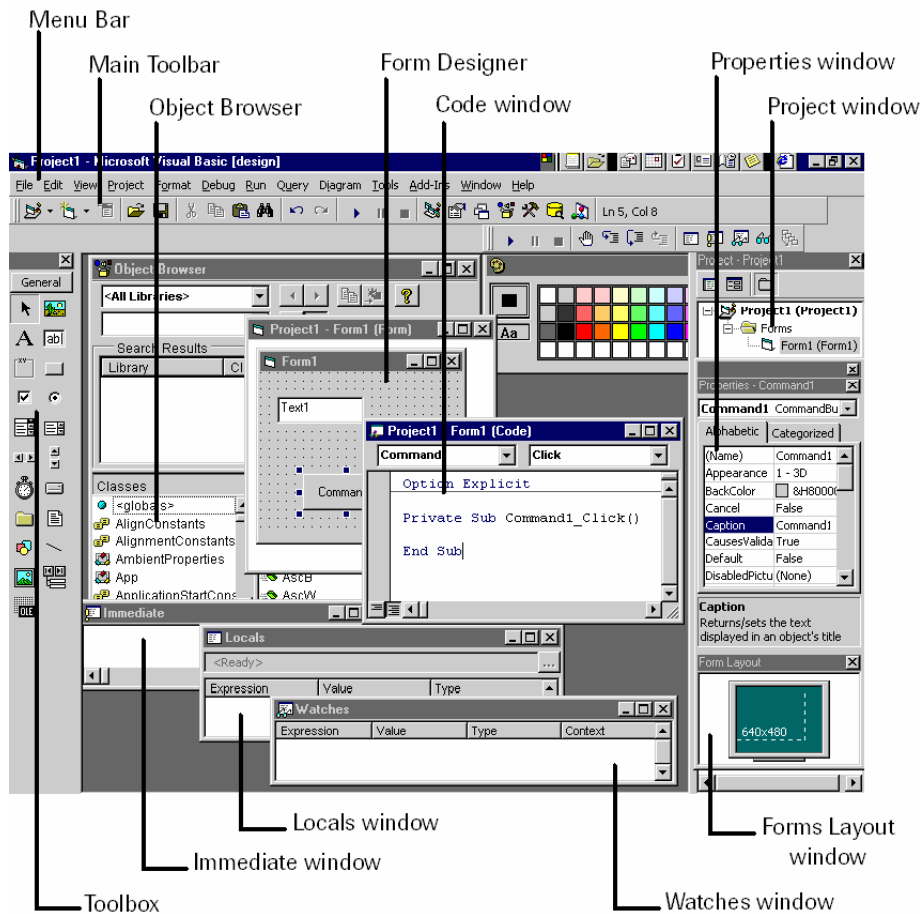
Gambar 7.2. Tampilan awal Visual Basic.

Pada Gambar 7.2 kita diminta untuk memilih jenis proyek yang akan kita buat. Untuk tahap awal proyek Standar.EXE merupakan pilihan yang biasa dilakukan. Setelah kita memilih Standar.EXE maka kita akan dibawa pada tampilan berikut.



Gambar 7.3. Tampilan awal untuk pilihan Standard.EXE.

Sebelum kita bekerja dengan Visual Basic, maka ada baiknya kita mengenali lebih dulu lingkungan kerja (IDE) Visual Basic. IDE (*Integrated Development Environment*) Visual Basic merupakan Lingkungan Pengembangan Terpadu bagi programmer dalam mengembangkan aplikasinya. Dengan menggunakan IDE programmer dapat membuat *user interface*, melakukan *coding*, melakukan *testing* dan *debuging* serta menkompilasi program menjadi *executable*. Penguasaan yang baik akan IDE akan sangat membantu *programmer* dalam mengefektifkan tugas-tugasnya sehingga dapat bekerja dengan efisien. Tampilan IDE VB dapat dilihat pada Gambar 7.4.

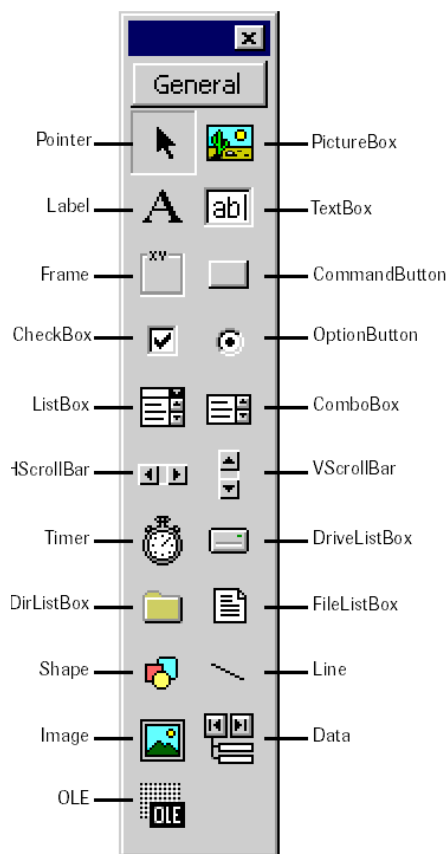


Gambar 7.4. IDE Visual Basic

- *Menu Bar*, digunakan untuk memilih tugas-tugas tertentu seperti menyimpan project, membuka project, dll
- *Main Toolbar*, digunakan untuk melakukan tugas-tugas tertentu dengan cepat.
- *Jendela Project*, jendela ini berisi gambaran dari semua modul yang terdapat dalam aplikasi anda. Anda dapat menggunakan *icon Toggle Folders* untuk menampilkan modul-modul dalam jendela tersebut secara di *group* atau berurut berdasarkan nama. Anda dapat menggunakan Ctrl+R untuk menampilkan jendela project, ataupun menggunakan icon *Project Explorer*.
- *Jendela Form Designer*, jendela ini merupakan tempat anda untuk merancang *user interface* dari aplikasi anda. Jadi jendela ini menyerupai kanvas bagi seorang pelukis.
- *Jendela Toolbox*, jendela ini berisi komponen-komponen yang dapat anda gunakan untuk mengembangkan *user interface*.

- *Jendela Code*, merupakan tempat bagi anda untuk menulis kode. Anda dapat menampilkan jendela ini dengan menggunakan kombinasi Shift-F7.
- *Jendela Properties*, merupakan daftar properti-properti *object* yang sedang terpilih. Sebagai contohnya anda dapat mengubah warna tulisan (*foreground*) dan warna latarbelakang (*background*). Anda dapat menggunakan F4 untuk menampilkan jendela properti.
- *Jendela Color Palette*, adalah fasilitas cepat untuk mengubah warna suatu *object*.
- *Jendela Form Layout*, akan menunjukkan bagaimana *form* bersangkutan ditampilkan ketika *run-time*.

Jendela *Toolbox* merupakan jendela yang sangat penting bagi anda. Dari jendela ini anda dapat mengambil komponen-komponen (*object*) yang akan ditanamkan pada *form* untuk membentuk *user interface*.



Gambar 7.5. Toolbox VB 6.

- *Pointer* bukan merupakan suatu kontrol; gunakan *icon* ini ketika anda ingin memilih kontrol yang sudah berada pada *form*.
- *PictureBox* adalah kontrol yang digunakan untuk menampilkan *image* dengan format: BMP, DIB (bitmap), ICO (icon), CUR (cursor), WMF (metafile), EMF (*enhanced metafile*), GIF, dan JPEG.
- *Label* adalah kontrol yang digunakan untuk menampilkan teks yang tidak dapat diperbaiki oleh pemakai.
- *TextBox* adalah kontrol yang mengandung string yang dapat diperbaiki oleh pemakai, dapat berupa satu baris tunggal, atau banyak baris.
- *Frame* adalah kontrol yang digunakan sebagai kontainer bagi kontrol lainnya.
- *CommandButton* merupakan kontrol hampir ditemukan pada setiap *form*, dan digunakan untuk membangkitkan *event* proses tertentu ketika pemakai melakukan klik padanya.

- *CheckBox* digunakan untuk pilihan yang isinya bernilai *yes/no*, *true/false*.
- *OptionButton* sering digunakan lebih dari satu sebagai pilihan terhadap beberapa option yang hanya dapat dipilih satu.
- *ListBox* mengandung sejumlah item, dan user dapat memilih lebih dari satu (bergantung pada property *MultiSelect*).
- *ComboBox* merupakan kombinasi dari *TextBox* dan suatu *ListBox* dimana memasukkan data dapat dilakukan dengan pengetikkan maupun pemilihan.
- *HScrollBar* dan *VScrollBar* digunakan untuk membentuk *scrollbar* berdiri sendiri.
- *Timer* digunakan untuk proses background yang diaktifkan berdasarkan interval waktu tertentu. Merupakan kontrol *non-visual*.
- *DriveListBox*, *DirListBox*, dan *FileListBox* sering digunakan untuk membentuk dialog box yang berkaitan dengan file.
- *Shape* dan *Line* digunakan untuk menampilkan bentuk seperti garis, persegi, bulatan, oval.
- *Image* berfungsi menyerupai image box, tetapi tidak dapat digunakan sebagai kontainer bagi kontrol lainnya. Sesuatu yang perlu diketahui bahwa kontrol image menggunakan resource yang lebih kecil dibandingkan dengan *PictureBox*
- *Data* digunakan untuk *data binding*
- *OLE* dapat digunakan sebagai tempat bagi program eksternal seperti Microsoft Excel, Word, dll.

#### 7.1.1. Prinsip Pokok Pemrograman Berbasis GUI

Secara prinsip ada dua bagian pokok dalam pengembangan aplikasi dengan menggunakan VB, yaitu: *visual design* dan *event-driven programming*.

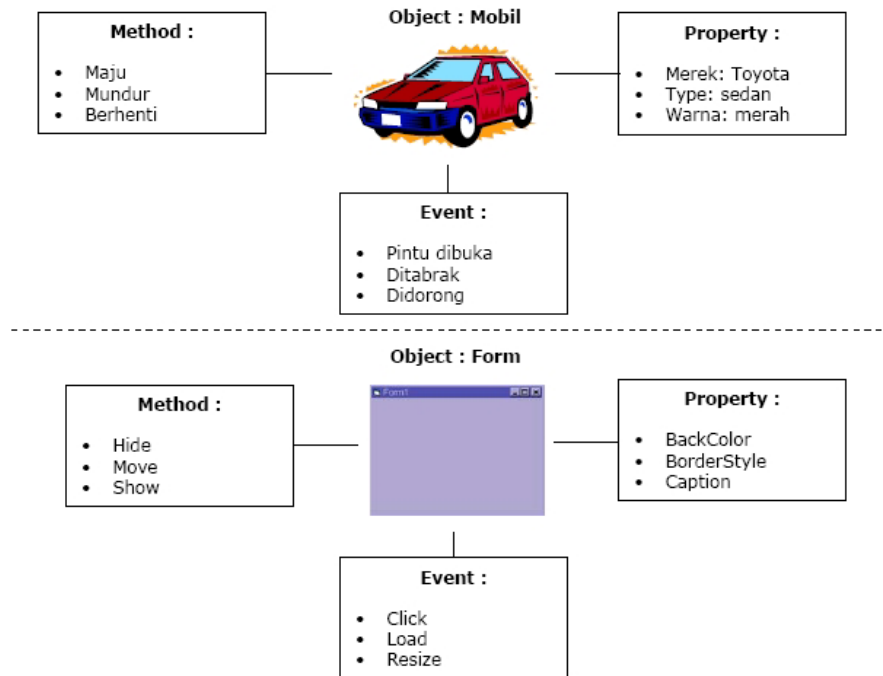
##### Visual Design

Dalam lingkungan Windows, *User-interface* sangat memegang peranan penting, karena dalam pemakaian aplikasi yang kita buat, pemakai senantiasa berinteraksi dengan *User-interface* tanpa menyadari bahwa dibelakangnya berjalan instruksi-instruksi program yang mendukung tampilan dan proses yang dilakukan. Pada pemrograman Visual, pengembangan aplikasi selalu dimulai dengan pembentukan *user interface*. Untuk mendisain *user interface*, pengetahuan yang paling dibutuhkan hanyalah pemahaman dasar tentang jenis dan kegunaan dari control dan dasar-dasar bagaimana menggambar sebuah *object*.

*Form* dan *control* merupakan elemen-elemen dasar dalam *user interface* pada aplikasi-aplikasi berbasis Windows. Dalam VB, elemen-elemen ini disebut obyek karena dapat dimanipulasi seperti sebuah obyek. Obyek merupakan

suatu kombinasi dari kode dan data yang dapat diperlakukan sebagai satu kesatuan. Sebuah obyek memiliki sejumlah *property* dan sejumlah *method*, dan akan bereaksi terhadap sejumlah *event* eksternal seperti halnya obyek fisik.

Sebagai ilustrasi (gambar 7.6), sebuah mobil adalah sebuah obyek fisik yang memiliki *property*, *method* dan *event*. Salah satu *property* adalah warna. Biasanya *property* warna dari mobil ditetapkan ketika sebuah mobil dibuat. Tetapi kalau kita tidak suka dengan warna mobil yang kita beli, kita masih dapat merubahnya, misalnya dengan mengecat ulang. Demikian juga dalam VB, *property* dari sebuah *control* biasanya ditentukan ketika *object* dibuat (pada saat ditempatkan pada sebuah *form*), tetapi kita dapat merubah *property* ini dengan memberikan nilai baru. Kita dapat merubah nilai *property* pada saat disain (dengan menggunakan jendela Properties) atau pada saat *runtime* (melalui kode program). Beberapa *property* hanya tersedia pada saat disain, dan beberapa *property* hanya tersedia pada saat *runtime*.



Gambar 7.6. Obyek, *Property*, *Method* dan *Event*

### Event-Driven Programming

Pemrograman suatu aplikasi bukanlah sesuatu yang mudah, namun ada sebuah metodologi yang tidak boleh kita tinggalkan. Aplikasi yang dibuat dengan VB bukanlah sebuah program yang monolithic (hanya ada satu urutan jalannya program aplikasi). Ketika kita membuat program dengan VB, pertama kita harus menentukan bagaimana aplikasi akan berinteraksi dengan pengguna.

Atau dengan kata lain, kita harus menentukan bagaimana setiap control bereaksi terhadap aksi yang dilakukan pengguna (misalnya: *click mouse*, *double-click mouse*, penekanan salah satu tombol pada *keyboard*, dan lain-lain). Konsep ini biasa disebut sebagai *Event-Driven Programming*, karena bukan aplikasi yang menentukan alur namun kejadian (*event*) yang disebabkan oleh pengguna yang menentukan alur dari aplikasi. Program aplikasi yang kita buat bereaksi terhadap kondisi eksternal (*event*), dan aksi dari pengguna yang menentukan bagaimana alur dari aplikasi.

### 7.1.2. Tipe Data, Variabel dan Konstanta

Secara umum tipe data, variabel dan konstanta dalam Visual Basic tidak banyak berbeda dengan apa yang disampaikan pada Bab 5. Coba buka kembali Bab tersebut untuk mengingat kembali bagian ini. Yang berbeda adalah bentuk pendeklarasian variabel dan konstanta. Perhatikan contoh berikut.

Contoh 7.1. Contoh pendeklarasian variabel, konstanta dan tipe data.

```
Dim speed As Double
Dim timeElapsed As Double
Dim NumberStudent as Integer = 10
Dim velocity as Single
Dim Nama as String
Const phi as Single = 3.14
```

Pada contoh 7.1 kita bisa mendeklarasikan variabel dengan kata kunci Dim sedangkan untuk konstanta menggunakan kata kunci Const. Kita juga bisa langsung mengisi nilai awal setelah definisi tipe data dari suatu variabel.

Salah satu keuntungan menggunakan pemrograman berbasis GUI seperti Visual Basic adalah tersedianya obyek-obyek yang dapat diperlakukan sebagai tipe data. Perhatikan contoh berikut ini.

Contoh 7.2. Contoh penggunaan tipe data obyek.

```
Dim frm As Form
Dim midfrm As MDIForm
Dim ctrl As Control
Dim obj As Object
Dim inv As frmInvoice
Dim txtSalary As TextBox
Dim wrk As Excel.Worksheet
```

Pada contoh 7.2 di atas, pernyataan Form, MDIForm, Control dan Object adalah kontrol-kontrol yang dimiliki oleh Visual Basic. Sedangkan frmInvoice adalah form yang telah kita beri nama frmInvoice. Visual Basic juga memungkinkan kita untuk menggunakan obyek dari luar Visual Basic. Perhatikan pada baris terakhir di atas. Kita menggunakan *worksheet* dari Excel untuk kita gunakan dalam program kita.



### 7.1.3. Operator

**Operator** adalah simbol yang digunakan dalam bahasa pemrograman untuk melakukan suatu operasi terhadap nilai data. Simbol operator bisa berupa karakter ataupun kata khusus. Pada Visual Basic ada tiga kelompok operator utama, yaitu operator aritmatika, operator perbandingan dan operator logika.

- **Operator aritmatika**

**Operator aritmatika** digunakan untuk operasi matematis terhadap nilai data. Notasi-notasi yang digunakan tampak pada tabel berikut ini :

Tabel 7.1. Operator Aritmatika.

Simbol	Operasi Matematis	Contoh
^	pemangkatan	5 ^ 2 hasilnya 25
*	perkalian	5 * 2 hasilnya 10
/	pembagian (hasil pecahan)	5 / 2 hasilnya 2,5
\	pembagian (hasil bulat)	5 \ 2 hasilnya 2
Mod	sisa pembagian	5 Mod 2 hasilnya 1
+	penjumlahan	5 + 2 hasilnya 7
-	pengurangan	5 - 2 hasilnya 3
&	penggabungan string	5 & 2 hasilnya 52

Notasi/simbol untuk operator ini memiliki hirarki kekuatan, artinya apabila ada dua atau lebih operator digunakan bersama-sama, maka operator dengan hirarki yang lebih tinggi akan dieksekusi lebih dulu. Hirarki kekuatan operator secara berturut-turut adalah ^, \* dan /, \, mod, + dan -. Operator \* dan / adalah setara. Demikian juga operator + dan -. Perhatikan contoh berikut ini.

Contoh 7.3. Contoh penggunaan hirarki operator aritmatika.

$$\begin{aligned}5 * 2 + 3 &= 13 \\4 ^ 2 - 5 &= 11\end{aligned}$$

Pada contoh pertama, tanda \* (kali) memiliki kekuatan yang lebih tinggi dari tanda + (tambah) sehingga operasi perkalian dilakukan terlebih dahulu daripada penjumlahan. Hasil yang diperoleh adalah 10 ditambah 3 bukan 5 dikali 5. Sedangkan pada contoh kedua, tanda ^ (pangkat) mempunyai urutan kekuatan lebih tinggi dari - (kurang) sehingga hasil akhirnya adalah 11 (yaitu dari 16 dikurangi 5). Untuk merubah urutan proses aritmatika dapat diatur dengan menggunakan tanda kurung. Perhatikan contoh berikut :

Contoh 7.4. Contoh penggunaan kurung untuk mengatur hirarki operator.

$$\begin{aligned}5 * (2 + 3) &= 25 \\4 ^ (2 - 5) &= 0.015625\end{aligned}$$

Pada contoh ini kita menggunakan angka dan operator yang sama seperti contoh sebelumnya, namun dengan menggunakan tanda kurung untuk merubah urutan proses. Pada contoh pertama proses yang pertama dilakukan adalah penjumlahan 2 dan 3, kemudian baru dikalikan 5. Hasil akhirnya adalah

25 (bandingkan dengan contoh sebelumnya, yaitu 13). Sedangkan pada contoh kedua proses pertama adalah 2 dikurangi 5 kemudian baru hasilnya digunakan sebagai angka pangkat untuk 4 (menjadi  $4^{(-3)}$ ). Hasil yang diperoleh adalah 0.015625, berbeda jauh dengan contoh sebelumnya yaitu 11. Dari kedua contoh ini dapat dilihat bahwa tanda kurung memiliki kekuatan lebih tinggi dari operator lainnya. Apabila ada lebih dari satu tanda kurung berjajar maka tanda kurung paling dalam yang akan diproses terlebih dahulu. Perhatikan contoh berikut :

$$5 * ((2 + 2) / 8) = 2.5$$

- **Operator perbandingan**

**Operator perbandingan** digunakan untuk operasi yang membandingkan nilai data. Simbol-simbol yang digunakan :

Tabel 7.2. Operator perbandingan.

Simbol	Operasi Perbandingan	Contoh
<	lebih kecil	$5 < 2$ hasilnya FALSE
>	lebih besar	$5 > 2$ hasilnya TRUE
<=	lebih kecil atau sama dengan	$5 <= 2$ hasilnya FALSE
>=	lebih besar atau sama dengan	$5 >= 2$ hasilnya TRUE
=	sama dengan	$5 = 2$ hasilnya FALSE
<>	tidak sama dengan	$5 <> 2$ hasilnya TRUE

- **Operator logika**

**Operator logika** digunakan untuk operasi yang membandingkan suatu perbandingan. Simbol-simbol yang digunakan :

Tabel 7.3. Operator logika.

Simbol	Operasi Logika	Contoh
Or	atau	$(5 < 2)$ Or $(5 > 2)$ hasilnya TRUE
And	dan	$(5 < 2)$ And $(5 > 2)$ hasilnya FALSE
Not	Tidak	Not $(5 < 2)$ hasilnya TRUE

#### 7.1.4. Struktur Kontrol Program

Struktur kontrol program atau kadang disebut sebagai struktur kendali merupakan penerapan dari algoritma struktur percabangan dan struktur pengulangan. Buka kembali Bab 5 untuk memperjelas kembali tentang struktur algoritma.

- Struktur percabangan.

Struktur percabangan dalam Visual Basic dapat dilakukan dengan menggunakan **If ... Then** dan **Select ... Case**. **If ... then** digunakan jika percabangan tidak terlalu banyak. Sedangkan **Select ... Case** digunakan jika ada banyak percabangan. Perhatikan contoh penulisan **If ... Then** ini.

Contoh 7.5. Contoh penulisan **If .. Then**.

' Percabangan/pemilihan satu baris tanpa Else  
**If x > 0 Then y = x**

' Percabangan/pemilihan satu baris dengan Else  
**If x > 0 Then y = x Else y = 0**

' Percabangan/pemilihan satu baris dengan Else dan titik dua  
**If x > 0 Then y = x: x = 0 Else y = 0**

' Percabangan ditulis dengan cara lebih dari satu baris  
**If x > 0 Then**  
     **y = x**  
     **x = 0**  
**Else**  
     **y = 0**  
**End If**

' Penggunaan blok if  
**If x > 0 Then**  
     **y = x**  
**ElseIf x < 0 Then**  
     **y = x \* x**  
**Else**  
     **x = -1**  
**End If**

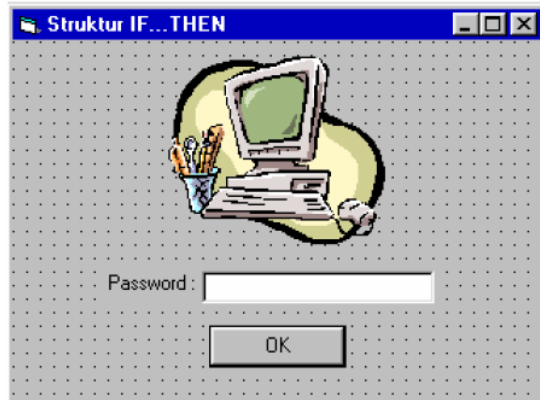
Contoh 7.6. Contoh penulisan **Select ... Case**.

```
Dim position As Integer    'Pilihan
position = CInt(txtPosition.Text)
Select Case position
    Case 1
        txtOutcome.Text = "Menang"
    Case 2
        txtOutcome.Text = "Kalah"
    Case 3
        txtOutcome.Text = "Seri"
    Case Else
        txtOutcome.Text = "Tidak bertanding."
End Select
```

Coba perhatikan contoh aplikasi dengan menggunakan If ... Then dan Select ... Case berikut ini.

#### Contoh 7.7. Program aplikasi dengan if ... then.

Aktifkan VB 6 kemudian buatlah form sebagai berikut :



Atur property untuk masing-masing obyek sebagai berikut :

Object	Properties	Value
Form5	Caption StartPosition	Struktur IF...THEN 2 – CenterScreen
Image1	Stretch Picture Visible	True Komputer.wmf False
Label1	Caption	Password :
Text1	PasswordChar Text	* <kosong>
Command1	Caption Default	OK True

Buka Jendela *Code* dan pada bagian Code Editor ketikkan kode programnya sebagai berikut :

```
Private Sub Command1_Click()  
    If Text1.Text = "nusantara" Then Image1.Visible =  
    True  
End Sub
```

Klik menu **Project > Project1 Properties** lalu klik tab General. Gantilah Startup Object-nya menjadi Form5. Coba jalankan Project1 :

- Ketikkan sembarang teks pada TextBox lalu klik tombol OK atau tekan Enter, tidak terjadi apa-apa.
- Ketikkan "nusantara" pada TextBox lalu klik tombol OK atau tekan Enter, gambar computer akan muncul.

Penjelasan kode program :

If Text1.Text = "nusantara" Then Image1.Visible = True

kondisi                      kode program yang dikerjakan  
bila kondisi TRUE

Modifikasi kode programnya menjadi sebagai berikut :

```
Private Sub Command1_Click()  
If Text1.Text = "nusantara" Then  
    Image1.Visible = True  
    Text1.Enabled = False  
    Command1.Enabled = False  
Else  
    MsgBox "Passwordnya Salah !"  
    Text1.Text = ""  
    Text1.SetFocus  
End If  
End Sub
```

Coba jalankan kembali Project1 :

- Ketikkan sembarang teks pada TextBox lalu klik tombol OK atau tekan Enter, muncul kotak pesan "Passwordnya Salah !". Klik tombol OK pada kotak pesan tersebut.
- Ketikkan "nusantara" pada TextBox lalu klik tombol OK atau tekan Enter, gambar computer akan muncul. TextBox dan tombol OK menjadi *disable* (tidak bisa digunakan).

Penjelasan kode program :

kondisi

If Text1.Text = "nusantara" Then  
 Image1.Visible = True  
 Text1.Enabled = False  
 Command1.Enabled = False  
Else  
 MsgBox "Passwordnya Salah !"  
 Text1.Text = ""  
 Text1.SetFocus  
End If

blok kode program yang dikerjakan  
bila kondisi TRUE

blok kode program yang dikerjakan  
bila kondisi FALSE

**Catatan tambahan :**

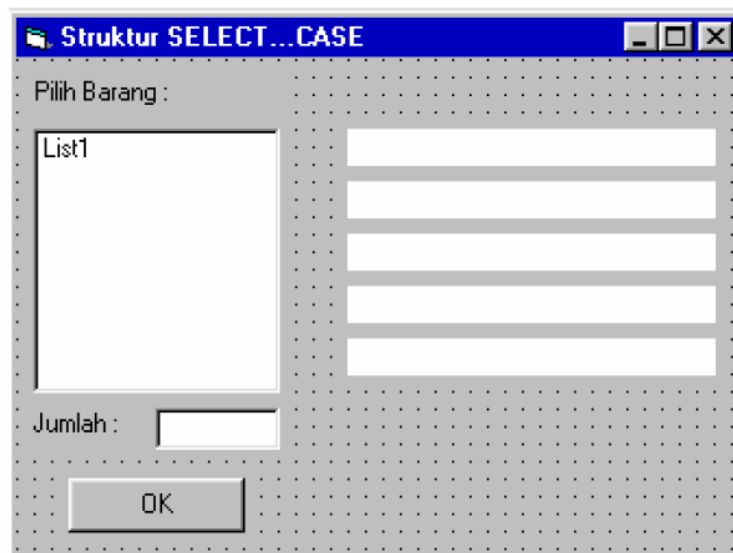
- Teks "nusantara" harus diketik huruf kecil semua. Ingat : data string bersifat *case sensitive* !
- Agar password-nya tidak bersifat *case sensitive*, modifikasi pernyataan kondisinya menjadi :

**If LCase(Text1.Text) = "nusantara" Then**

Fungsi LCase adalah untuk mengkonversi semua string yang diinput ke Text1.Text menjadi huruf kecil, walaupun user menginputnya dengan huruf kapital.

**Contoh 7.8. Program aplikasi dengan Select ... Case.**

Aktifkan VB 6 melalui tombol Start, kemudian buat form sebagai berikut :



Atur property untuk masing-masing obyek sebagai berikut :

Object	Properties	Value
Form6	Caption StartPosition	Struktur SELECT...CASE 2 – CenterScreen
Label1	Caption	Pilih Barang :
List1	-	-
Label2	Caption	Jumlah :
Text1	Text	<kosong>
Command1	Caption	OK
Label3-7	Name  BackColor Caption	lblBarang, lblHarga, lblJumlah, lblDiskon, lblTotal Palette : <putih> <kosong>

Buka Jendela Code dan pada bagian Code Editor ketikkan kode programnya sebagai berikut :

```
Private Sub Form_Load()  
    List1.AddItem "Disket"  
    List1.AddItem "Buku"  
    List1.AddItem "Kertas"  
    List1.AddItem "Pulpen"  
End Sub  
  
Private Sub Command1_Click()  
    Dim harga As Currency, total As Currency  
    Dim jumlah As Integer  
    Dim diskon As Single  
    Dim satuan As String  
    If List1.Text = "" Then  
        MsgBox "Anda belum memilih barang !!"  
        List1.ListIndex = 0  
        Exit Sub  
    End If  
    If Text1.Text = "" Then  
        MsgBox "Anda belum mengisi jumlah barang !!"  
        Text1.SetFocus  
        Exit Sub  
    End If  
    Select Case List1.Text  
        Case "Disket"  
            harga = 35000  
            satuan = "Box"  
        Case "Buku"  
            harga = 20000  
            satuan = "Lusin"  
        Case "Kertas"  
            harga = 25000  
            satuan = "Rim"  
        Case "Pulpen"  
            harga = 10000  
            satuan = "Pak"  
    End Select  
  
    lblBarang.Caption = "Barang : " & List1.Text  
    lblHarga.Caption = "Harga : " & Format(harga,  
    "Currency") & "/" & satuan  
    lblJumlah.Caption = "Jumlah : " & Text1.Text & " " &  
    satuan  
    jumlah = Text1.Text  
    Select Case jumlah  
        Case Is < 10  
            diskon = 0  
        Case 10 To 20  
            diskon = 0.15  
        Case Else
```

```

        diskon = 0.2
    End Select
    total = jumlah * (harga * (1 - diskon))
    lblDiskon.Caption = "Diskon : " & Format(diskon, "0
    %")
    lblTotal.Caption = "Total Bayar : " & Format(total,
    "Currency")
End Sub

```

Klik menu Project > Project1 Properties lalu klik tab General. Gantilah Startup Object-nya menjadi Form6.

Coba jalankan Project1 :

- List1 akan terisi nama-nama barang.
- Klik tombol OK, akan muncul kotak pesan "Anda belum memilih barang !!".
- Klik OK, nama barang pertama akan terpilih secara otomatis. Anda boleh memilih nama barang yang lainnya.
- Klik tombol OK, akan muncul kotak pesan "Anda belum mengisi jumlah barang !!".
- Klik OK, text1 akan menjadi focus. Isi jumlah barangnya, misalnya : 10.
- Klik tombol OKm akan tampil : nama barang, harga barang (persatuannya), jumlah barang (dengan satuannya), diskon dan total bayar.
- Coba ganti pilihan nama barang (pada List1) dan jumlah barang (pada Text1) lalu klik lagi tombol OK.



Penjelasan kode program :

```
Select Case List1.Text      cek barang yang dipilih :
Case "Disket"
    harga = 35000
    satuan = "Box" }      bila barang = Disket
Case "Buku"
    harga = 20000
    satuan = "Lusin" }    bila barang = Buku
Case "Kertas"
    harga = 25000
    satuan = "Rim" }      bila barang = Kertas
Case "Pulpen"
    harga = 10000
    satuan = "Pak" }      bila barang = Pulpen
End Select

lblBarang.Caption = "Barang : " & List1.Text
lblHarga.Caption = "Harga : " & Format(harga, "Currency") & "/" & satuan }      tampilkan has
lblJumlah.Caption = "Jumlah : " & Text1.Text & " " & satuan
```

jumlah = Text1.Text

```
Select Case jumlah      cek jumlah barang :
Case Is < 10
    diskon = 0      bila jumlah < 10 → diskon = 0%
Case 10 To 20
    diskon = 0.15    bila jumlah 10-20 → diskon = 15%
Case Else
    diskon = 0.2      bila jumlah > 20 → diskon = 20%
End Select

total = jumlah * (harga * (1 - diskon))

lblDiskon.Caption = "Diskon : " & Format(diskon, "0 %")
lblTotal.Caption = "Total Bayar : " & Format(total, "Currency") }      Hitung total bayar dan
                                                                    tampilkan hasilnya
```

#### Catatan :

- o Bila jumlah barang diisi dengan selain angka akan muncul pesan error.
- o Untuk mengecek isi Text1 angka atau bukan, tambahkan kode berikut :

```
If Not IsNumeric(Text1.Text) Then
    MsgBox "Isi jumlah barang harus angka !!"
    Text1.SetFocus
Exit Sub
End If
```

- Struktur pengulangan.

Struktur pengulangan yang mungkin paling banyak digunakan dalam Visual Basic adalah For. Dalam Visual Basic, struktur for ini dikenal sebagai For ... Next. Cara penulisan umum For ... Next adalah sebagai berikut:

```
For counter = nilaiAwal To nilaiAkhir [Step increment]
    ' pernyataan yang akan diulang...
Next
```

Pernyataan Step dan increment sudah disinggung di Bab 5 sehingga tidak akan dibahas lagi di sini. Perhatikan contoh penggunaan For ... Next dalam aplikasi berikut ini.

Coba perhatikan contoh pengulangan dengan For ... Next berikut ini.

Contoh 7.9. Pengulangan dengan For ... Next.

```
Dim d As Single, count As Long
For d = 0 To 10 Step 2
    count = count + 1
Next
Print count
```

Pada Contoh 7.9, d adalah counter dan kita deklarasikan sebagai single. Kita juga menggunakan increment dengan nilai 2. Nilai increment dapat berupa bilangan bulat atau pecahan. Namun nilai pecahan terkadang memberikan hasil yang tidak kita inginkan. Bagaimanakah output dari program di atas? Pada akhir program maka nilai count akan sama dengan 5.

Struktur pengulangan yang lebih fleksibel dari *For ... Next* adalah *Do ... Loop*. *Do ... Loop* dapat berbentuk berbeda-beda. Perhatikan contoh berikut.

Contoh 7.10. Pengulangan dengan *For ... Next*.

```
Do While x > 0
    y = y + 1
    x = x \ 2
Loop

Do
    y = y + 1
    x = x \ 2
Loop Until x <= 0
```

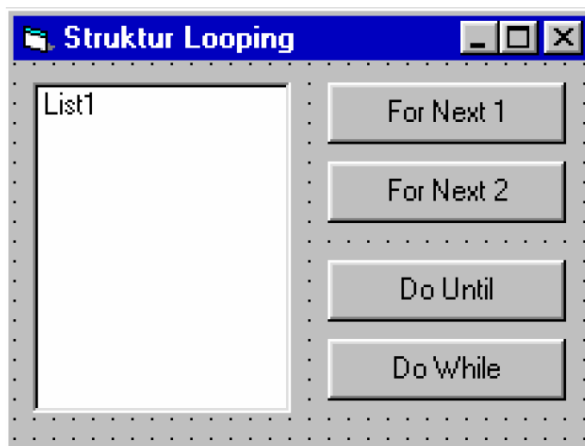
Pada bagian pertama dari Contoh 7.10 kita menggunakan *Do While ... Loop*. Cara ini sama persis dengan struktur pengulangan dengan While yang telah kalian pelajari pada Bab 5. Pernyataan di bawah *Do While*, akan dikerjakan jika kondisi pada *Do While* bernilai benar. Jika tidak maka tidak akan dieksekusi.

Bagian kedua dari Contoh 7.10, kita menggunakan *Do ... Loop Until* untuk melakukan pengulangan. Pada bentuk seperti ini, pengulangan dilakukan sampai kondisi pada *Loop Until* bernilai benar. Jadi selama kondisi di depan *Loop until* masih bernilai salah maka pengulangan akan terus dilakukan. Atau *Do ... Loop* ini merupakan kebalikan dari *Do While*. Melihat kedua bagian tersebut, apabila kita inisialisasi  $x = -4$  dan  $Y = 5$ , bagaimanakah hasilnya? Pada pengulangan dengan *Do While*, tidak akan memberikan hasil karena nilai  $X$  kurang dari 0 sehingga pengulangan tidak dilakukan. Sedangkan pada *Do ... Loop Until*, pernyataan di bawah *Do* masih dikerjakan, karena kondisi baru diperiksa di akhir pengulangan. Sehingga nilai  $X$  akan bernilai -2 dan  $Y$  bernilai 6.

Berikut ini adalah contoh program aplikasi dengan menggunakan struktur pengulangan.

Contoh 7.11. Program aplikasi dengan struktur pengulangan.

Buka VB dan buatlah form sebagai berikut :



Atur property seperti tabel berikut :

Object	Properties	Value
Form7	Caption StartPosition	Struktur Looping 2 – CenterScreen
List1	-	-
Command1-4	Caption	For Next 1 For Next 2 Do Until Do While

Buka Jendela Code dan pada bagian Code Editor ketikkan kode programnya sebagai berikut :

```
Dim i As Integer

Private Sub Command1_Click()
List1.Clear
```

```

For i = 1 To 100
    List1.AddItem "Angka " & i
Next i
End Sub

Private Sub Command2_Click()
List1.Clear
For i = 100 To 1 Step -2
    List1.AddItem "Angka " & i
Next i
End Sub

Private Sub Command3_Click()
List1.Clear
i = Asc("A")
Do Until i > Asc("Z")
    List1.AddItem "Huruf " & Chr(i)
    i = i + 1
Loop
End Sub

Private Sub Command4_Click()
List1.Clear
i = Asc("Z")
Do While i >= Asc("A")
    List1.AddItem "Huruf " & Chr(i)
    i = i - 1
Loop
End Sub

```

Coba jalankan program. Tekan keempat tombol yang ada pada form bergantian. Cobalah cermati output apa yang keluar dari eksekusi program di atas.

#### 7.1.5. Prosedur dan Fungsi

Ada beberapa jenis procedure yang digunakan dalam Visual Basic :

- Sub procedure yang tidak mengembalikan nilai
- Function procedure yang mengembalikan nilai
- Property procedure yang dapat mengembalikan nilai dan diisi nilai yang mengacu pada suatu objek.

- **Sub procedure**

Syntax penulisan Sub procedure:

```
[Private|Public][Static]Sub namaprosedur (argumen-argumen)  
    pernyataan-pernyataan  
End Sub
```

Setiap kali *procedure* dipanggil, maka *pernyataan-pernyataan* yang berada di antara Sub dan End Sub akan dijalankan. Argumen pada *procedure* adalah nilai yang akan dilewatkan saat pemanggilan *procedure*.

Di Visual Basic *Sub Procedure* dapat dibagi atas dua yaitu :

- **General Procedure**, *procedure* yang diaktifkan oleh aplikasi.
- **Event Procedure**, *Procedure* yang diaktifkan oleh system sebagai respon terhadap event.

Contoh 7.12. Contoh *sub procedure*.

Pada contoh ini kita akan membuat sub procedure dengan nama CenterForm yang dapat digunakan untuk menampilkan form ketengah Layar, dimana x adalah parameter yang merupakan form yang akan dibuat ketengah layar.

```
Sub CenterForm(x As Form)  
    x.Top = (Screen.Height - x.Height) \ 2  
    x.Left = (Screen.Width - x.Width) \ 2  
End Sub  
  
'memanggil sub prosedur CenterForm  
Private Sub Form_Load()  
    Call CenterForm(Me)  
End Sub
```

Pada contoh 7.12, *sub procedure* CenterForm membutuhkan argumen form. Sehingga ketika dipanggil argumen juga harus dicantumkan. Perhatikan pada baris Call CenterForm(Me). Me di sini adalah argumen dari sub procedure. Pada Visual Basic, Me merujuk pada form dimana kode program ini dibuat.

- **Function procedure**

Ada dua jenis function dalam Visual Basic, yaitu *Built-in Function* dan *Function Procedure*. Pada Visual Basic telah disediakan banyak *Built-in Function* yang dapat digunakan untuk berbagai tujuan seperti fungsi-fungsi untuk perhitungan matematika, manipulasi *string*, manipulasi tipe data dan lain-lain. Pada bab ini tidak akan dibahas detil fungsi-fungsi built-in tersebut. Namun pada lampiran, dapat dilihat beberapa fungsi built-in yang sering digunakan.

Meskipun sudah tersedia sangat banyak, tetapi fungsi-fungsi yang tersedia tersebut bersifat umum dan kadang-kadang tidak memenuhi

kebutuhan programmer, untuk keperluan tersebut anda dapat menciptakan fungsi-fungsi sendiri yang dikenal dengan **Function procedure**. Atau kita bisa mengatakan sebagai fungsi buatan sendiri.

Sintaks penulisan *function procedure* :

```
[Private|Public][Static]Function      namaprocedure
(argumen-argumen) [As type]
    statements
End Function
```

Ada tiga perbedaan antara *function* dan *procedure* :

- o Umumnya anda dapat memanggil suatu *function* dengan mengikutkan nama *function* sisi kanan dari statement atau ekspresi. (returnvalue = function()).
- o *Function* memiliki type data seperti suatu variabel. Ini menentukan type yang dari nilai yang dikembalikan.
- o Nilai kembali dimasukkan ke namafunction itu sendirinya, dan suatu function dapat menjadi bagian dari suatu ekspresi yang panjang.

Perhatikan contoh fungsi berikut ini.

#### Contoh 7.13. Contoh Fungsi.

Fungsi ini adalah fungsi yang bekerja untuk menampilkan nama bulan dalam bahasa Indonesia dari data tanggal yang dimasukkan. Argumen yang dibutuhkan dalam fungsi ini adalah x dan bertipe data *date*.

```
Function Bulan(x As Date)
Dim sRet As String

Select Case Month(x)
    Case 1: sRet = "Januari"
    Case 2: sRet = "Februari"
    Case 3: sRet = "Maret"
    Case 4: sRet = "April"
    Case 5: sRet = "Mei"
    Case 6: sRet = "Juni"
    Case 7: sRet = "Juli"
    Case 8: sRet = "Agustus"
    Case 9: sRet = "September"
    Case 10: sRet = "Oktober"
    Case 11: sRet = "Nopember"
    Case 12: sRet = "Desember"
    Case Else
        sRet = "tidak sah"
    End Select

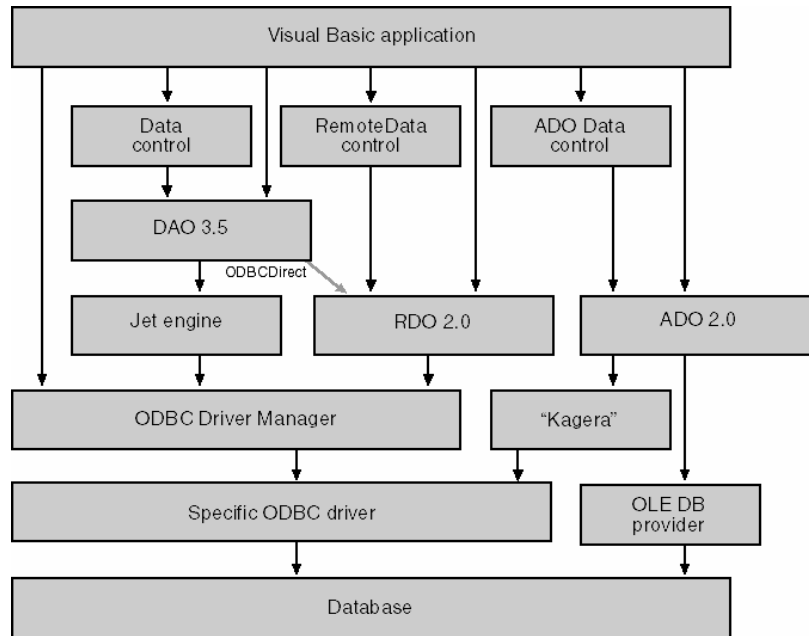
Bulan = sRet

End Function
```

## 7.2. Akses dan Manipulasi Basis Data dengan Visual Basic

Salah satu keunggulan Visual Basic dibandingkan dengan Bahasa pemrograman yang lain, adalah kemampuan dalam akses dan manipulasi data. Hal ini karena Visual Basic dikembangkan oleh Microsoft yang juga membuat sistem operasi Windows. Sehingga fungsi-fungsi basis data mendapat dukungan yang sangat lengkap.

Visual Basic menyediakan beragam cara untuk mengakses dan memanipulasi data. Perhatikan diagram pada gambar berikut ini.



Gambar 7.7. Berbagai cara akses basis data dari Visual Basic.

- ODBC

ODBC singkatan dari *Open Data Base Connectivity*. Merupakan sekumpulan fungsi yang membolehkan kita untuk koneksi pada basis data lokal atau yang berada di jaringan. Biasanya ODBC digunakan untuk mengakses berbagai tipe basis data antara lain, MS FoxPro, MS Access, MS SQL Serve, Oracle atau bahkan data yang tersimpan dalam bentuk file data.

- DAO

DAO atau *Data Access Object* adalah antarmuka dari Microsoft Jet, perangkat utama penggerak MS Access. Kita dapat membuat basis data dengan MS Access kemudian kita manipulasi dengan Visual Basic melalui DAO

dengan mudah. Karena DAO langsung berhubungan dengan MS Access, maka kita dapat menggunakan fungsi-fungsi DBMS dari Visual Basic.

- OLE DB

OLE DB adalah teknologi untuk mengakses basis data level dasar yang sebenarnya dimaksudkan untuk menggantikan fungsi ODBC. Namun dalam perkembangannya ODBC dan OLE DB memiliki perbedaan, yaitu OLE DB berbasis pada teknologi COM dan dapat digunakan untuk mengakses basis data yang bukan bersifat relasional.

- ADO

ADO (*ActiveX Data Object*) adalah antarmuka level atas dari OLE DB. ADO dikembangkan di atas OLE DB untuk melengkapi fungsi-fungsi yang tidak dimiliki oleh OLE DB sehingga memudahkan programmer dalam membuat aplikasi.

### 7.2.1. Membuat dan Manipulasi Basis Data dengan ADO

Secara prinsip, apapun metode akses yang digunakan, cara bekerja dengan basis data dari Visual Basic adalah sama. Dibutuhkan beberapa tahapan untuk dapat bekerja dengan basis data. Pada bagian ini kita akan menggunakan ADO sebagai teknologi yang lebih fleksibel dari teknologi yang lain. Tapi ini dapat dikembangkan dengan teknologi yang lain.

- Koneksi dengan basis data.

Koneksi dengan basis data berarti kita menghubungkan basis data supaya terbuka dan bisa kita akses data yang ada didalamnya. Berikut contoh pernyataan untuk koneksi ke basis data biblio.mdb

```
Dim cn As New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.3.51;"
—
    & "Data Source=C:\Microsoft Visual
Studio\Vb98\Biblio.mdb"
```

- Mengakses record data pada basis data..

Kumpulan dari *record-record* data dalam basis data, pada ADO disebut sebagai *recordset*. Untuk bekerja dengan *recordset* ini kita membutuhkan tabel atau *view/query* yang ada dalam basis data. Perhatikan contoh berikut ini.

```
Const DBPATH = "C:\Program Files\Microsoft Visual
Studio\Vb98\NWind.mdb"
Dim cn As New ADODB.Connection, rs As New ADODB.Recordset
cn.Open "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=" &
DBPATH
```



```
rs.Source = "Employees"  
rs.Open , cn
```

Pada kode di atas kita menggunakan basis data Nwind.mdb sebagai Data source. Kemudian sebagai sumber *recordset*, kita memanggil tabel "Employees". Variabel rs adalah *recordset*. Setelah kita tentukan sumber tabelnya, kita dapat membuka tabel tersebut dengan pernyataan open.

Setelah terbuka tabel sumbernya, kita dapat mengakses data yang ada di dalamnya. Berikut ini contoh kode untuk mengakses data pada suatu tabel.

```
Dim i As Integer  
For i = 0 To rs.Fields.Count  
    Print rs.Fields(i).Name & " = " & rs.Fields(i).Value  
Next
```

Perintah di atas akan mencetak seluruh baris dan kolom Name yang ada pada tabel *Employees* yang telah terbuka sebelumnya. Cara yang paling cepat adalah dengan menggunakan perintah *For Each* seperti kode berikut.

```
Dim fld As ADODB.Field  
For Each fld In rs.Fields  
    Print fld.Name & " = " & fld  
Next
```

- Manipulasi data dalam *recordset*

Untuk mengupdate data pada suatu *recordset* dapat dilakukan dengan cara sebagai berikut:

```
rs.Update Array("FirstName", "LastName", "BirthDate",  
"HireDate"), _  
    Array("John", "Smith", #1/1/1961#, #12/3/1994#)
```

Sedangkan untuk menambah *recordset* perintah yang digunakan adalah sebagai berikut:

```
rs.AddNew  
rs("FirstName") = "Robert"  
rs("LastName") = "Doe"  
rs("BirthDate") = #2/5/1955#  
rs.Update
```

Menghapus *record* tertentu dapat dilakukan dengan perintah seperti berikut.

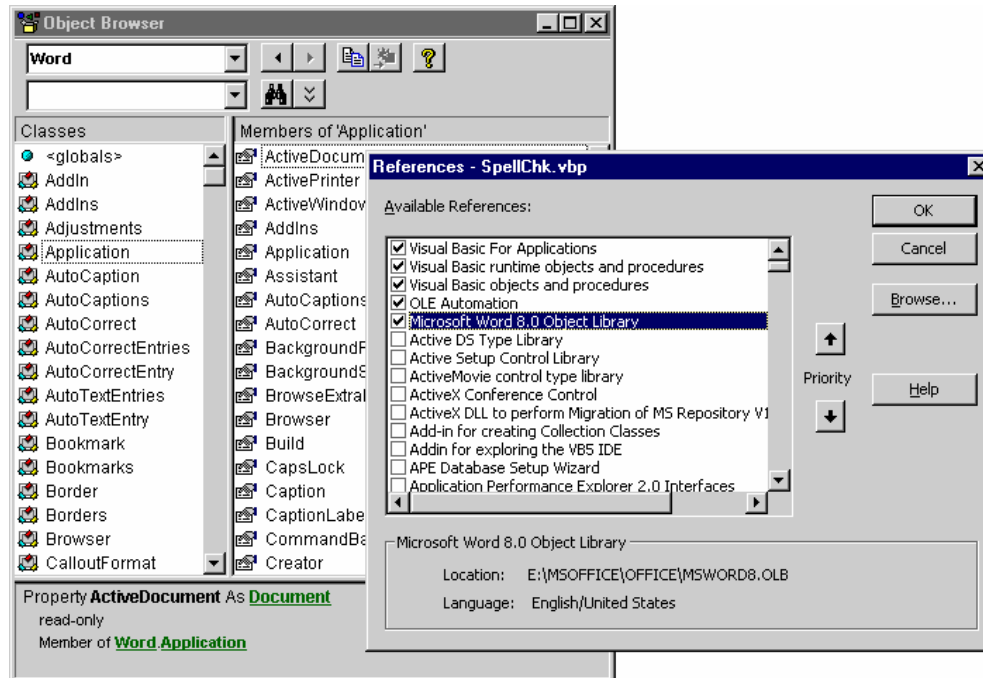
```
rs.Delete  
rs.MoveNext  
If rs.EOF Then rs.MoveLast
```

### 7.3. Teknologi COM

#### 7.3.1. Pengertian dan Konsep COM

COM atau *Component Object Model* adalah infrastruktur yang disediakan oleh Visual Basic untuk mengakses obyek-obyek atau kontrol-kontrol lain sepanjang punya antar muka yang dapat diakses oleh Visual Basic.

Untuk dapat menggunakan COM pada Visual Basic, kita dapat membuka **Reference Dialog** dengan cara: dari menu **Project** pilih **add Reference**. Jendela seperti pada Gambar 7.8 akan terbuka.



Gambar 7.8. Jendela Reference.

Pada Gambar 7.8 kita memilih COM atau obyek yang ingin kita gunakan. Pada contoh ini kita memilih "Microsoft Word 8.0 Object Library". Setelah kita klik OK maka kita dapat menggunakan obyek yang kita pilih ini dalam kode program kita. Berikut contoh penggunaan obyek tersebut.

#### Contoh 7.14. Contoh penggunaan COM.

```
Private Sub cmdCheck_Click()  
    Dim text As String  
    Dim suggestion As Word.SpellingSuggestion  
    Dim colSuggestions As Word.SpellingSuggestions
```

```

' menambahkan dokumen bila belum ada dokumen yang
terbuka.
If MSWord.Documents.Count = 0 Then MSWord.Documents.Add
text = Trim$(txtWord.text)

lstSuggestions.Clear
If MSWord.CheckSpelling(text) Then
    lstSuggestions.AddItem "(correct)"
Else
    Set colSuggestions =
MSWord.GetSpellingSuggestions(text)
    If colSuggestions.Count = 0 Then
        lstSuggestions.AddItem "(no suggestions)"
    Else
        For Each suggestion In colSuggestions
            lstSuggestions.AddItem suggestion.Name
        Next
    End If
End If
End Sub

```

Pada kode di atas kita menggunakan salah satu metode dari obyek yang kita buka sebelumnya ("Microsoft Word 8.0 Object Library"). Metode yang ingin kita gunakan adalah **SpellingSuggestion** (lihat kode pada bagian pendeklarasian variabel). Dengan cara yang sama kita dapat menggunakan metode atau fungsi-fungsi yang kita inginkan dari obyek COM yang telah kita muat.

Pada jendela *Reference* di Gambar 7.8, tersedia banyak sekali COM yang dapat kita gunakan. Cobalah untuk membuka satu persatu dan lihat apa fungsinya.

#### 7.4. Ringkasan

Pada bab ini kalian telah mempelajari pemrograman dengan Visual Basic. Dimulai dengan penerapan variabel, konstanta, tipe data dan operator. Kemudian dilanjutkan dengan strukturkendali pemrograman baik percabangan maupun pengulangan. Penggunaan procedure dan function juga disampaikan untuk melengkapi dasar pemrograman dengan Visual Basic.

Pada bagian selanjutnya kalian juga telah mempelajari teknik akses dan manipulasi data dengan menggunakan ADO. Bagian akhir ditutup dengan bagaimana membuka dan menggunakan teknologi COM yang disediakan oleh Visual Basic.

### 7.5. Soal-Soal Latihan

1. Berapakah hasil dari ekspresi Visual Basic berikut ini:
  - a.  $3*4$
  - b.  $7^2$
  - c.  $1/(2^3)$
  - d.  $3 + (4*5)$
  - e.  $(5 - 3)*4$
  - f.  $3*((-2)^5)$
2. Berapakah hasil dari operasi menggunakan mod berikut ini:
  - a.  $6 \text{ Mod } 2$
  - b.  $14 \text{ Mod } 4$
  - c.  $7 \text{ Mod } 3$
  - d.  $5 \text{ Mod } 5$
3. Periksa salah penamaan variabel dalam Visual Basic ini. Tentukan mana yang diperbolehkan dan yang tidak.
  - a. `sales.2006`
  - b. `room&Board`
  - c. `fOrM_1040`
  - d. `1040B`
  - e. `expenses?`
  - f. `INCOME 2006`
4. Jika  $a = 2$ ,  $b = 3$ , dan  $c = 4$ , berapakah hasil dari ekspresi berikut ini.
  - a.  $(a*b) + c$
  - b.  $a*(b + c)$
  - c.  $(1 + b)*c$
  - d.  $a^c$
  - e.  $b^(c - a)$
  - f.  $(c - a)^b$
5. Buatlah program untuk menghitung ekspresi berikut ini.
  - a.  $7*8 + 5$
  - b.  $(1 + 2*9)^3$
  - c.  $5.5\% \text{ of } 20$
  - d.  $15 - 3(2 + 3^4)$
  - e.  $17(3 + 162)$
  - f.  $4 \frac{1}{2} - 3 \frac{5}{8}$
6. Bukalah Gambar 5.6 dan 5.7 pada Bab 5. Buatlah program dalam Visual Basic. Gunakan kontrol TextBox dan CommandButton pada soal ini.
7. Bukalah Gambar 5.9 dan 5.10 pada Bab 5. Buatlah program dalam Visual Basic.
8. Bukalah Gambar 5.15 dan 5.17 pada Bab 5. Buatlah program dalam Visual Basic.

---

**BAB 8 PEMROGRAMAN BERORIENTASI OBYEK DENGAN JAVA**


---



Gambar 8.1. Logo Java.

Gambar cangkir dengan uap mengepul di atasnya serta tulisan Java seperti Gambar 8.1 ini mungkin pernah kalian lihat. Ya, ini adalah logo dari bahasa pemrograman Java yang popularitasnya meningkat beberapa tahun belakangan ini. Banyak *game* dan aplikasi yang digunakan pada perangkat *mobile* seperti telepon selular dan PDA dibuat dengan bahasa ini. Bahasa ini dikenal karena portabilitas dan dukungan pada konsep pemrograman berorientasi obyek.

Bab ini meliputi dua standar kompetensi, yaitu membuat program dalam bahasa pemrograman berorientasi obyek dan membuat program aplikasi menggunakan Java. Hal ini karena kedekatan konsep antara Java dan pemrograman berorientasi obyek. Standar kompetensi membuat program dalam bahasa pemrograman berorientasi obyek terdiri dari empat kompetensi dasar yaitu tipe data dan kontrol program, pembuatan kelas, penggunaan inheritance, polymorphism, dan overloading, dan penggunaan interface dan paket. Sedangkan standar kompetensi membuat program aplikasi menggunakan Java terdiri dari lima kompetensi dasar, yaitu menjelaskan file I/O, tipe data dan variabel, menerapkan operator, menjelaskan exception handling, menerapkan multi-threading dan menjelaskan network programming.

Dalam penyajian pada buku ini, satu sub bab tidak langsung merujuk pada satu kompetensi dasar. Ringkasan diletakkan pada setiap akhir bab kemudian dilanjutkan dengan soal-soal latihan. Sebelum mempelajari kompetensi ini ingatlah kembali sistem operasi, prinsip pemecahan masalah, algoritma pemrograman dasar dan lanjutan, pemrograman dengan VB dan VB.Net dan materi-materi pendukung dari mata pelajaran lain.

## TUJUAN

Setelah mempelajari bab ini diharapkan pembaca akan mampu :

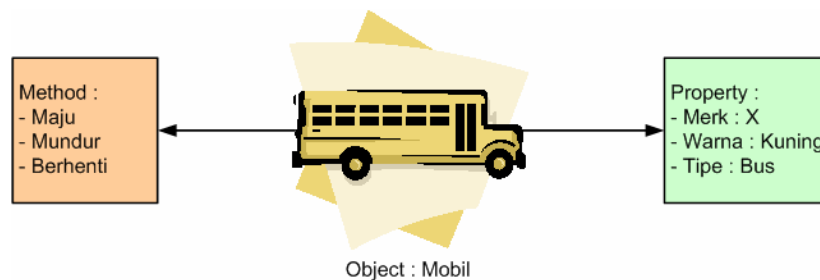
- o Memahami konsep pemrograman berorientasi obyek
- o Menjelaskan File I/O, tipe data dan variabel pada Java
- o Menggunakan operator
- o Menerapkan kontrol program
- o Menjelaskan Exception Handling
- o Menerapkan Multi-Threading
- o Menjelaskan Network Programming
- o Membuat program berorientasi obyek dengan *class*
- o Menggunakan prinsip inheritance, polymorphism dan overloading
- o Membuat program berorientasi obyek dengan interface dan paket

### 8.1. KONSEP PEMROGRAMAN BERORIENTASI OBYEK

Pemrograman Berorientasi Obyek (*Object Oriented Programming – OOP*) adalah *programming paradigm* yang menggunakan obyek dan interaksinya untuk merancang aplikasi dan program komputer. OOP tidak banyak digunakan sebelum awal tahun 1990an. Tapi sekarang menjadi sesuatu yang sudah lumrah digunakan. Bahasa-bahasa pemrograman seperti keluarga dotNet dari Microsoft (Visual Basic.Net, Visual C#, dan Visual J), Borland Delphi, Java, Phyton, PHP versi 5 ke atas, C++ dan banyak lainnya merupakan bahasa pemrograman yang mendukung konsep OOP.

Apakah obyek itu? Semua benda yang ada didunia ini dapat kita sebut sebagai obyek. Guru mata pelajaran RPL kalian adalah suatu obyek. Buku yang kalian pegang ini juga suatu obyek. Bahkan mata pelajaran RPL adalah juga sebuah obyek. Setiap obyek akan mempunyai karakteristik dan tingkah laku tertentu. Karakteristik disebut *attribute* dan tingkah laku disebut sebagai *behavior* atau *method*.

Dalam difinisi pemrograman berorientasi obyek dikenal adanya kelas dan obyek. *Class* atau kelas mendefinisikan karakteristik abstrak dari sesuatu termasuk atribut atau sifat-sifat dari sesuatu dan apa yang dapat dikerjakan oleh sesuatu (*method*). Sebagai contoh, mobil adalah sebuah kelas yang memiliki attribut warna, merek, tipe dan lain-lain. Mobil juga punya *method* antara lain, maju, mundur dan berhenti (lihat Gambar 8.2).



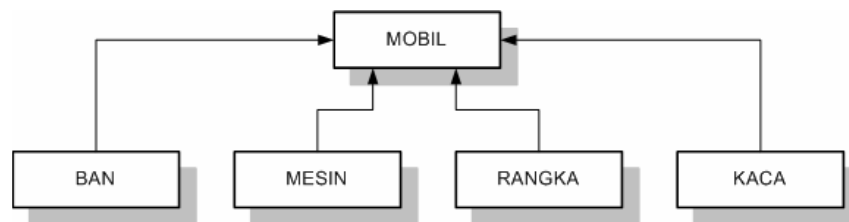
Gambar 8.2. Kelas, atribut dan *method*.

Obyek adalah contoh dari kelas yang sudah didefinisikan. Atribut dan method dari kelas secara otomatis akan menurun pada obyek namun dengan kekhususan. Sebagai ilustrasi kita perhatikan Gambar 8.2. Pada gambar tersebut, kita bisa identifikasi kelasnya adalah mobil dengan atribut dan methodnya. Obyeknya adalah sebuah mobil sedan dengan merk Toyota, dan warnanya adalah merah. Sedan itu juga memiliki *method* maju, mundur dan berhenti. Pada kasus ini mobil sedan disebut sebagai *instance* atau turunan dari kelas mobil.

Ada beberapa konsep penting yang kalian harus pahami dalam pemrograman berorientasi yaitu, abstraksi, enkapsulasi, *inheritance* dan *polymorphism*.

### 8.1.1. Abstraksi

*Abstraction* atau disebut juga *composition* merupakan prinsip penyederhanaan dari sesuatu yang kompleks dengan cara memodelkan kelas sesuai dengan masalahnya. Untuk lebih memperjelas pengertian coba perhatikan Gambar 8.3. Pada gambar tersebut terlihat sebuah mobil jika dipecah-pecah bagian-bagiannya kita akan dapatkan seperti ban, mesin, rangka mobil, kaca, dan lain-lain dan hal ini berlaku sebaliknya. Jika kita gabungkan bagian-bagian tersebut maka kita akan mendapatkan sebuah kelas mobil. Pada pemrograman berorientasi obyek biasanya kalau kita menjumpai beberapa kelas atau obyek yang kalau diidentifikasi memiliki banyak kesamaan atribut dan method maka kita akan menggabungkan kelas-kelas tersebut menjadi satu *super class*.



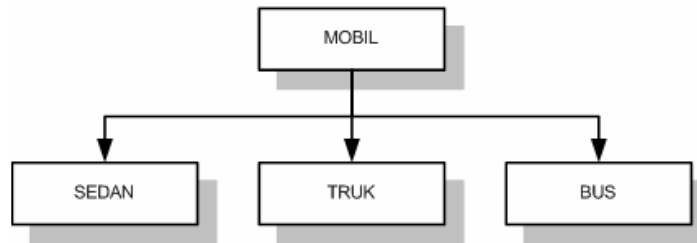
Gambar 8.3. Contoh abstraction.

### 8.1.2. Enkapsulasi

Prinsip *encapsulation* adalah prinsip menyembunyikan detail dari sebuah kelas terhadap obyek yang berinteraksi terhadapnya. Sebagai contoh ketika kita menjalankan mobil, sebenarnya kita sedang berinteraksi dan kita meminta kepada mobil untuk menjalankan *methodnya* seperti maju, mundur atau berhenti. Kita berinteraksi hanya dengan beberapa bagian dari mobil (*interface*) seperti persneling, setir, pijakan gas, pijakan rem dan bagian lain. Tapi detail proses yang terjadi didalam mobil bagaimana bisa maju, mundur atau berhenti kita tidak perlu tahu.

### 8.1.3. Inheritance

*Inheritance* atau pewarisan adalah prinsip pewarisan sifat dari orang tua ke anak atau turunannya yang diterapkan pada kelas. Orang tua memiliki atribut dan method yang lebih umum dibandingkan dengan anak atau turunannya. Pada Gambar 8.4. kita bisa tahu bahwa mobil memiliki atribut dan method yang lebih umum dibandingkan dengan sedan , truk atau bus. Mobil sebagai kelas yang mewarisi disebut sebagai *super class*, sedangkan sedan, truk dan bus sebagai kelas yang diwarisi disebut *sub class*.



Gambar 8.4. Pewarisan.

### 8.1.4. Polymorphism

*Polymorphism* mungkin merupakan konsep pemrograman beroorientasi obyek yang paling sulit dimengerti. Arti dari polymorphism adalah kemampuan dari suatu obyek untuk mempunyai lebih dari satu bentuk. Atau dalam pengertian lain adalah kita dapat menerapkan sesuatu hal yang berbeda melalui suatu cara yang sama. Sebagai contoh kalau ada empat ekor hewan berbeda yaitu burung, ular, katak, dan singa kemudian kita minta untuk bergerak, maka burung akan terbang, ular akan melata, katak melompat, singa mungkin akan berlari. Jadi suatu method yang sama mungkin bisa diterapkan secara lain jika obyek yang menerapkan adalah berlainan.

## 8.2. PENGENALAN PADA JAVA

Bahasa pemrograman Java pertama lahir dari The Green Project, yang berjalan selama 18 bulan, dari awal tahun 1991 hingga musim panas 1992. Proyek tersebut menggunakan versi yang dinamakan Oak. Nama Oak ini tidak dipakai untuk versi release Java karena sebuah perangkat lunak sudah terdaftar dengan merek dagang tersebut, sehingga diambil nama penggantinya menjadi "Java". Nama ini diambil dari kopi murni yang digiling langsung dari biji (kopi tubruk). Saat ini Java berada dibawah lisensi Sun Microsystems

Menurut definisi dari Sun, Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Orang orang yang berkecimpung dalam dunia pemrograman lebih suka menyebut Java sebagai sebuah teknologi



dibanding hanya sebuah bahasa pemrograman, karena Java lebih lengkap dibanding sebuah bahasa pemrograman konvensional.

### 8.2.1. Kebutuhan Perangkat Lunak

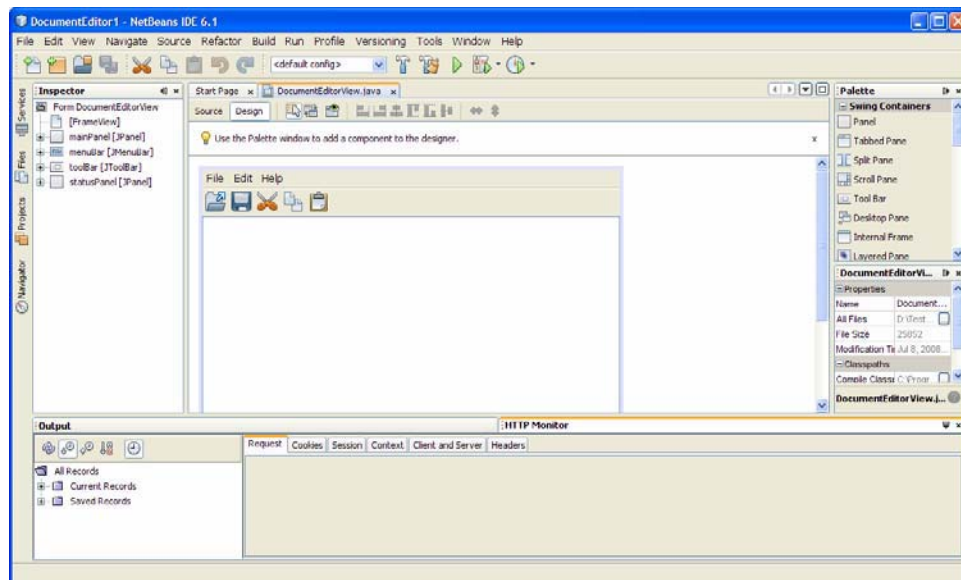
Untuk membuat program Java paling tidak harus tersedia dua buah software yaitu:

- **Java 2 SDK Standard Edition (J2SE).**

Perangkat lunak ini merupakan yang akan kita gunakan untuk mengkompilasi kode program Java yang kita buat. Selain itu pada perangkat lunak ini tersedia kelas-kelas yang dapat kita gunakan untuk membangun aplikasi desktop, grafis, keamanan, konektivitas basis data dan jaringan. Perangkat lunak ini dapat di-*download* gratis di situs Sun Microsystem. Setelah itu perangkat lunak ini harus diinstal pada system operasi yang kita pakai.

- **Text Editor.**

Perangkat lunak ini berfungsi untuk menuliskan kode-kode program. Notepad, Vi, Gedit, merupakan contoh-contoh teks editor yang dapat digunakan. Namun menggunakan teks editor agak menyulitkan karena tidak tersedia fasilitas bantu yang memudahkan dalam menuliskan kode program. Saat ini beberapa IDE tersedia gratis di internet. IDE tersebut telah menyediakan banyak fasilitas seperti *syntax coloring*, *auto completion*, dan *template* untuk memudahkan membuat aplikasi berbasis Java. NetBeans ([www.netbeans.org](http://www.netbeans.org)) dan Eclipse ([www.eclipse.org](http://www.eclipse.org)) merupakan dua buah IDE yang sangat terkenal dan sangat kuat. Gambar 8.5. menunjukkan tampilan NetBeans IDE.



### 8.2.2. Kompilasi Kode Program

Sebelum kita membuat program dan mengkompilasinya kita harus mengatur **ClassPath**. **ClassPath** adalah suatu sistem variabel yang digunakan untuk mengatakan pada program yang ditulis dengan bahasa Java tempat lokasi-lokasi yang akan digunakan. Misalkan kita meletakkan kode program kita di direktori **d:\TestCode\Java**, maka kita perlu mengatur path agar kita bisa menggunakan *compiler* java dari direktori ini. Berikut ini langkah-langkah membuat **classpath**:

- Buka *command-prompt* di Windows kemudian setelah terbuka ketikkan :  
**Set PATH=C:\progra~1\java\jdk1.5.0\bin;%PATH%**  
**Set CLASSPATH=.;D:\TestCode\Java**  
*Path* di atas adalah jika kita menggunakan JDK versi 1.5, jika versi yang lain maka tinggal menyesuaikan.
- Periksa apakah setting sudah benar dengan mengetikkan perintah java pada direktori manapun dari command prompt.

Setelah *classpath* terbentuk, coba buka Notepad kemudian ketikkan kode berikut ini.

```
public class Main {  
    //isi blok  
    public static void main(String[] args) {  
        System.out.println("Hallo ini Java lho");  
    }  
}
```

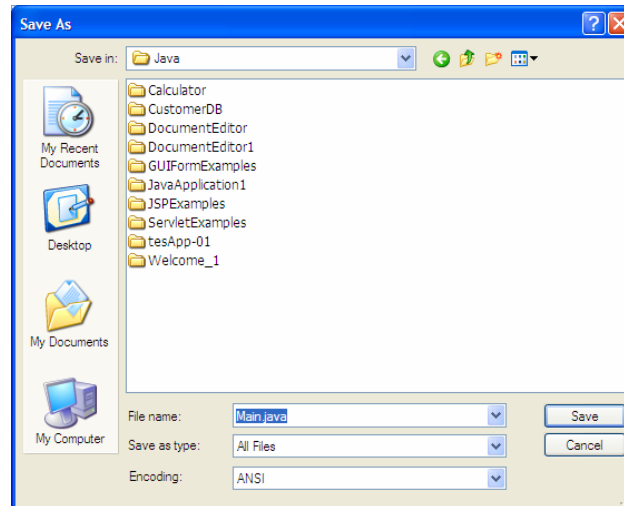
Kemudian simpan file kode tersebut dengan nama yang sama dengan *class* yang ada pada kode tersebut yaitu *Main* dan diakhiri dengan ekstensi **.java**. Simpan di lokasi yang telah kita tetapkan di atas (lihat Gambar 8.6). Aturan penulisan nama file di Java sangat ketat, sehingga jika kalian member nama file tidak sama dengan nama kelasnya, maka program akan menolak untuk dieksekusi.

Pada kode program di atas, baris pertama menunjukkan nama *class* yaitu *Main*. Pada Java semua kode seharusnya ditempatkan didalam deklarasi *class*. Kita melakukannya dengan menggunakan kata kunci *class*. Sebagai tambahan, *class* menggunakan akses khusus *public*, yang menunjukkan bahwa class kita mempunyai akses bebas ke *class* yang lain dari *package* yang lain pula (*package* merupakan kumpulan *class-class*).

Setelah deklarasi *class* kemudian diikuti tanda { yang menunjukkan awal blok kode. Tanda ini harus ditutup dengan tanda } sebagai akhir blok.

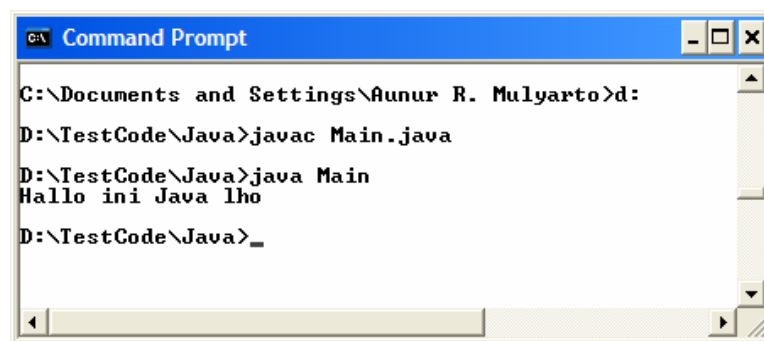
Baris yang dimulai dengan tanda // adalah komentar. Kemudian baris berikutnya adalah deklarasi nama *method*. Dalam hal ini nama *method*-nya

adalah *main* (*method* utama). Kita dapat membuat *method-method* lainnya diluar *main* ini. Setelah deklarasi ini diikuti juga dengan blok kode. Pada blok kode berisi pernyataan **System.out.println("Hallo ini Java lho");**. Perintah `System.out.println()`, menampilkan teks yang diapit oleh tanda petik ganda (" ") pada layar.



Gambar 8.6. Nama file dan lokasi penyimpanan.

Sekarang buka *command prompt* dan arahkan ke direktori tempat file java kalian simpan dan ketikkan seperti pada Gambar 8.7. Perhatikan cara penulisan dan hasil yang diperoleh.



Gambar 8.7. Cara eksekusi program dalam Java.

Sebelum dapat dieksekusi, maka kode program (**Main.java**) harus dikompilasi dengan menggunakan perintah **javac** seperti terlihat di Gambar 8.7. Setelah kompilasi berhasil, maka program dapat kita eksekusi dengan

menggunakan perintah **java**. Proses kompilasi akan menghasilkan file dengan nama **Main.class** (periksa direktori kalian dengan perintah **dir**, dan cari file dengan akhiran **.class**). Yang kita panggil pada perintah **java** adalah file dengan akhiran **class** ini dan bukan lagi kode sumber yang berakhiran **.java**.

### 8.3. TIPE DATA, VARIABEL, DAN PERNYATAAN INPUT/OUTPUT (I/O)

#### 8.3.1. Tipe Data

Ada 8 tipe data dasar pada Java yaitu *boolean* (untuk bentuk logika), *char* (untuk bentuk tekstual), *byte*, *short*, *int*, *long* (*integral*), *double* and *float* (*floating point*). Tabel 8.1 berikut menunjukkan penjelasan tentang tipe data tersebut.

Tabel 8.1. Tipe data pada Java.

Tipe Data	Penjelasan
logika ( <i>boolean</i> )	diwakili oleh dua pernyataan : true dan false
tekstual ( <i>char</i> )	harus memiliki ciri berada dalam tanda single quotes(' ')
<i>Integral</i> ( <i>byte</i> , <i>short</i> , <i>int</i> & <i>long</i> )	Tipe data bilangan bulat, default tipe data yaitu int. byte = 8 bits dengan range $-2^7$ s/d $2^7-1$ short = 16 bits dengan range $-2^{15}$ s/d $2^{15}-1$ int = 32 bits dengan range $-2^{31}$ s/d $2^{31}-1$ long = 64 bits dengan range $-2^{63}$ s/d $2^{63}-1$
<i>Floating Point</i> ( <i>float</i> dan <i>double</i> )	Tipe data bilangan asli (boleh ada pecahan). Default tipe datanya adalah double. float = 32 bits dengan range $-2^{31}$ s/d $2^{31}-1$ double= 64 bits dengan range $-2^{63}$ s/d $2^{63}-1$

Pada Java, *String* bukan merupakan tipe data primitif (namun merupakan suatu *Class*). *String* mewakili tipe data yang terdiri atas beberapa karakter. *String* ditulis dengan menggunakan tanda petik ganda ("").

Contoh-contoh berikut menunjukkan bagaimana menggunakan tipe-tipe data di atas. Ketikkan kode pada contoh-contoh berikut kemudian kompilasi dan jalankan. Perhatikan output dari program.

#### Contoh 8.1. Penggunaan tipe data integral.

```
public class ContohPerhitungan {
    public static void main(String[] args) {
        byte a = 1;
        short b = 12;
        int c = 300, d, e;
        d = a + b + c;
        e = a * b * c;
        System.out.println("Hasil penjumlahan = " + d);
        System.out.println("Hasil perkalian = " + e);
    }
}
```

```
}
}
```

Contoh 8.2. Penggunaan tipe data float.

```
public class LuasLingkaran {
    public static void main(String[] args) {
        double pi = 3.1416;
        double r = 2.12;
        double luas;
        luas = pi * r * r;
        System.out.println("Luas Lingkaran = " + luas);
    }
}
```

Contoh 8.3. Penggunaan tipe data char.

```
public class tipeChar {
    public static void main(String[] args) {
        char ch = 'A';
        System.out.println("ch = " + ch);
        ch++;
        System.out.println("ch = " + ch);
    }
}
```

### 8.3.2. Variabel dan Konstanta

Aturan penamaan (*identifier*) variabel dan konstanta seperti terdapat pada Bab 5 juga berlaku pada Java. Selain itu penulisan *identifier* pada Java bersifat *case-sensitive*. Artinya huruf besar dan huruf kecil dianggap suatu yang berbeda. Tidak seperti pada VB, Java mensyaratkan kita mendeklarasikan variabel dan konstanta lebih dulu. Kalau tidak maka kode program tidak akan dapat dikompilasi.

Cara pendeklarasian variabel adalah sebagai berikut:

```
<tipe data> <nama variabel> [=nilai awal];
```

Nilai awal bersifat opsional atau boleh dicantumkan atau tidak. Perhatikan contoh 8.1 di atas. Variabel a, b, dan c telah ditentukan nilai awalnya. Sedangkan variabel d dan e tidak ditentukan nilainya. Perhatikan juga contoh 8.2 dan 8.3 untuk pendeklarasian variabel.

### 8.3.3. Input / Output

Pada contoh-contoh kode program di atas sebenarnya kita telah menggunakan salah satu cara untuk menampilkan output ke layar, yaitu perintah **System.out.println**, namun kita belum pernah menggunakan pernyataan

untuk mendapatkan input. Berikut ini kita akan pelajari bagaimana menggunakan pernyataan input dan output pada Java.

Untuk dapat menangkap input dari keyboard, maka kita harus menggunakan kelas **BufferedReader** yang berada di *java.io package*. Sehingga di awal program kita harus mencantumkan kelas tersebut pada kode program. Perhatikan contoh berikut.

#### Contoh 8.4. Pernyataan input pada Java.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class InputKeyboard
{
    public static void main( String[] args ){
        BufferedReader dataIn = new BufferedReader(new
            InputStreamReader( System.in) );
        String name = "";

        System.out.print("Ketikkan nama anda:");

        try{
            name = dataIn.readLine();
        }catch( IOException e ){
            System.out.println("Error!");
        }
        System.out.println("Hai " + name + "!");
    }
}
```

Tiga baris yang diawali dengan perintah import menunjukkan bahwa kita akan menggunakan kelas **BufferedReader**, **InputStreamReader** dan **IOException** yang berada di *java.io package*. Penjelasan tentang *package* akan kita bahas pada bagian lain dari bab ini.

Pada statement,

```
BufferedReader dataIn = new BufferedReader(new
InputStreamReader( System.in) );
```

kita mendeklarasikan sebuah variabel bernama **dataIn** dengan tipe kelas **BufferedReader**. Kemudian, kita mendeklarasikan variabel *String* dengan *identifier name*. Pernyataan ini digunakan untuk menunjukkan tempat menyimpan input dari pengguna. Variabel name diinisialisasi sebagai *String* kosong "". Baris berikutnya adalah memberikan output *string* seperti yang telah kita pelajari dengan menggunakan pernyataan `System.out.print`;

Sekarang, blok di bawah ini merupakan *try-catch* blok (kita akan bahas tentang ini di sub bab *exception*)

```
try{
    name = dataIn.readLine();
}catch( IOException e ){
    System.out.println("Error!");
}
```

Pada baris ini menjelaskan bahwa kemungkinan terjadi error pada statement `name = dataIn.readLine();` akan ditangkap. Jika ada kesalahan maka tulisan "Error" akan disampaikan. Jika tidak ada kesalahan maka variabel `name` akan diisi dengan apa yang dimasukkan oleh pengguna melalui *keyboard*. Dan akan ditampilkan pada pernyataan terakhir.

Untuk menampilkan output yang kita inginkan, dapat menggunakan perintah sebagai berikut :

```
System.out.println()
System.out.print()
```

`System.out.println()` akan membuat baris baru sedangkan `System.out.print()` tidak akan membuat baris baru.

## 8.4. OPERATOR

### 8.4.1. Operator Aritmatika

Operator aritmatika yang digunakan pada Java hampir sama dengan yang digunakan pada VB. Hanya pada penggunaan operator *modulus* yang berbeda notasinya. VB menggunakan `mod` sedangkan Java menggunakan tanda `%`. Tabel 8.2 menunjukkan operator aritmatika yang ada di Java.

Tabel 8.2. Operator aritmatika pada Java

Operator	Fungsi	Contoh
+	Penjumlahan	$3 + 5 = 8$
-	Pengurangan	$7 - 2 = 5$
*	Perkalian	$5 * 2 = 10$
/	Pembagian	$6 / 3 = 2$
%	Sisa hasil bagi (modulus)	$5 / 2 = 1$
++	Menambahkan nilai 1 ke variabel (increment)	$C++ = C + 1$
--	Mengurangi nilai 1 ke variabel	$C-- = C - 1$

Contoh berikut menunjukkan bagaimana menggunakan operator aritmatika. Ketikkan kode pada contoh berikut kemudian kompilasi dan jalankan. Perhatikan output dari program.

#### Contoh 8.4. Penggunaan operator aritmatika.

```
public class DemoAritmatika
{
    public static void main(String[] args)
    {
        int i = 21;
        int j = 38;
        double x = 9.123;
        double y = 12.78;
        //Cetak nilai variabel
        System.out.println("Nilai Variabel...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    x = " + x);
        System.out.println("    y = " + y);
        //penjumlahan
        System.out.println("Penjumlahan...");
        System.out.println("    i + j = " + (i + j));
        System.out.println("    x + y = " + (x + y));
        //pengurangan
        System.out.println("Pengurangan...");
        System.out.println("    i - j = " + (i - j));
        System.out.println("    x - y = " + (x - y));
        //perkalian
        System.out.println("Perkalian...");
        System.out.println("    i * j = " + (i * j));
        System.out.println("    x * y = " + (x * y));
        //pembagian
        System.out.println("Pembagian...");
        System.out.println("    i / j = " + (i / j));
        System.out.println("    x / y = " + (x / y));
        //modulus
        System.out.println("Sisa Hasil Bagi...");
        System.out.println("    i % j = " + (i % j));
        System.out.println("    x % y = " + (x % y));
        //increment
        System.out.println("Increment...");
        System.out.println("    i++ = " + (i++));
        System.out.println("    ++i = " + (++i));
        System.out.println("    j++ + i = " + (j++ + i));
        System.out.println("    ++j + i = " + (++j + i));
    }
}
```

#### 8.4.2. Operator Relasional

Operator relasional atau perbandingan pada Java juga mirip dengan VB. Hanya berbeda pada perbandingan symbol sama dan tidak sama. Pada VB untuk membandingkan dua buah operand apakah sama atau tidak menggunakan



operator = untuk sama dengan dan <> untuk tidak sama dengan. Sedangkan pada java digunakan == untuk sama dengan dan != untuk tidak sama dengan. Ketikkan kode pada contoh berikut kemudian kompilasi dan jalankan. Perhatikan output dari program.

Contoh 8.5. Penggunaan operator relasional.

```
public class DemoRelasional
{
    public static void main(String[] args) {
        int i = 20;
        int j = 16;
        int k = 16;
        //Cetak nilai variabel
        System.out.println("Nilai variabel...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    k = " + k);
        //lebih besar dari
        System.out.println("Lebih besar dari...");
        System.out.println("    i > j = " + (i > j));
        System.out.println("    j > i = " + (j > i));
        System.out.println("    k > j = " + (k > j));
        //lebih besar atau sama dengan
        System.out.println("Lebih    besar    atau    sama
dengan...");
        System.out.println("    i >= j = " + (i >= j));
        System.out.println("    j >= i = " + (j >= i));
        System.out.println("    k >= j = " + (k >= j));
        //lebih kecil dari
        System.out.println("Lebih kecil dari...");
        System.out.println("    i < j = " + (i < j));
        System.out.println("    j < i = " + (j < i));
        System.out.println("    k < j = " + (k < j));
        //lebih kecil atau sama dengan
        System.out.println("Lebih    kecil    atau    sama
dengan...");
        System.out.println("    i <= j = " + (i <= j));
        System.out.println("    j <= i = " + (j <= i));
        System.out.println("    k <= j = " + (k <= j));
        //sama dengan
        System.out.println("Sama dengan...");
        System.out.println("    i == j = " + (i == j));
        System.out.println("    k == j = " + (k == j));
        //tidak sama dengan
        System.out.println("Tidak sama dengan...");
        System.out.println("    i != j = " + (i != j));
        System.out.println("    k != j = " + (k != j));
    }
}
```

### 8.4.3. Operator Logika

.Ada 3 operator logika yang disediakan Java yaitu: && (AND), || (logika OR), | dan ! (logika NOT). Penggunaan operator ini sama persis dengan yang ada pada VB. Hanya notasinya saja yang berbeda.

## 8.5. STRUKTUR KONTROL PROGRAM

Seperti halnya pada VB, pada Java juga menyediakan struktur kontrol program untuk pemilihan maupun pengulangan. Perintah yang digunakan dalam struktur kontrol ini juga hampir mirip.

### 8.5.1. Struktur Pemilihan

Struktur pemilihan dapat menggunakan *if*, *if ... else*, dan *if ...else ... if*. Hal ini tidak berbeda jauh dengan apa yang telah kalian pelajari pada Bab 5 dan Bab 7. Perhatikan potongan kode-kode program berikut.

Contoh 8.6. Penggunaan if.

```
int nilai = 68;

if( nilai > 60 ) System.out.println("Selamat anda lulus!");
```

Contoh 8.6 ini hanya menggunakan if untuk membuat struktur pemilihan. Jika nilai lebih dari 60 maka program akan menampilkan output "Selamat anda lulus!"

Contoh 8.7. Penggunaan if ... else.

```
int nilai = 68;

if( nilai > 60 )      System.out.println("Selamat anda
lulus!");
else                  System.out.println("Anda tidak lulus!");
```

Pada contoh 8.7, kita menggunakan struktur if ... else. Jika nilai lebih dari 60 maka akan menampilkan output "Selamat anda lulus!" tetapi jika tidak (else) maka program akan menampilkan output "Anda tidak lulus!".

Contoh 8.8. Penggunaan if ... else ... if.

```
int nilai = 68;

if( nilai > 90 ){
    System.out.println("Nilai anda sangat baik!");
```

```

}
else if( nilai > 60 ){
    System.out.println("Nilai anda baik!");
}
else{
    System.out.println("Anda tidak lulus");
}

```

Contoh 8.8 merupakan pengembangan dari contoh 8.7. Jika nilai lebih dari 90 maka program akan menampilkan output "Nilai anda sangat baik!", tetapi jika kurang dari 90 dan lebih dari 60 (else if) maka program akan menampilkan output "Nilai anda baik!" dan jika tidak kedua-duanya (else) maka program akan menampilkan output "Anda tidak lulus"

Struktur pemilihan juga memungkinkan kita untuk memilih banyak alternatif. Namun jika menggunakan if akan sangat kompleks. Java menyediakan perintah switch. Perintah ini sama fungsinya dengan Select .. case pada VB. Perhatikan contoh berikut.

#### Contoh 8.9. Penggunaan switch.

```

public class SwitchControl {
    public static void main(String[] args) {
        int a=2;
        int b;
        switch(a) {
            case 1:
                b = a + 1;
                break;
            case 2:
                b = a + 2;
                break;
            case 3:
                b = a + 3;
                break;
            case 4:
                b = a + 4;
                break;
            default: b = 0;
        }
        System.out.println("Nilai b: " + b);
    }
}

```

Berapakah hasil dari kode program pada contoh 8.9 di atas. Kalau jawaban kalian 4 berarti kalian telah memahami bagaimana cara kerja perintah switch. Switch akan memeriksa apakah ada case yang memiliki nilai sama dengan a yaitu 2. Pemeriksaan dimulai dari case pertama yaitu 1. Pernyataan break harus dituliskan untuk menghentikan pencarian pada case berikutnya. Cobalah hilangkan pernyataan *break* dan jalankan kode program di atas. Bagaimanakah hasilnya?

### 8.5.2. Struktur Pengulangan

Ada tiga bentuk struktur pengulangan pada Java yaitu *for*, *while* dan *do-while*. Secara prinsip bentuk pengulangan ini sama dengan apa yang telah kalian pelajari pada Bab 5 dan 7. Berikut ini contoh-contoh bentuk pengulangan.

Contoh 8.10. Penggunaan *for* pada Java.

```
public class ForLoop {
    public static void main(String[] args) {
        int j=4;
        for (int x=0; x < 5; x++) {
            System.out.println("Nilai x: " + x);
            System.out.println("Nilai j: " + j);
            System.out.println();
            j--;
        }
    }
}
```

Sintaks umum *for* adalah : **for (nilai awal; kondisi; increment)** kemudian dilanjutkan dengan bagian yang akan diulang. Perhatikan baik-baik bagaimana menggunakan struktur *for* pada contoh 8.10 di atas. Nilai *x=0* adalah nilai awal. Sedangkan *x < 5* adalah kondisi yang harus dipenuhi agar pengulangan bisa dilakukan. Nilai *x++* merupakan *increment*. Ingat bahwa penulisan *x++* sama artinya dengan *x = x + 1*. Cobalah jalankan kode program di atas. Menurut kalian bagaimanakah keluaran dari program tersebut?

Contoh 8.11. Penggunaan *while* pada Java.

```
public class WhileLoop {
    public static void main(String[] args) {
        int y = 4;
        while ( y > 0 ){
            System.out.print(y);
            y--;
        }
    }
}
```

Pada contoh 8.11 ini kita menggunakan *while* untuk membuat pengulangan. Pada *while* kita perlu menginisialisasi variabel sebelum masuk ke bagian *while*. Variabel *y* kita inisialisasi dengan nilai 4. Kondisi yang harus dipenuhi pada *while* adalah *y>0*. Pada contoh ini counternya bersifat menurun (perhatikan bagian *y--*). Sehingga yang akan tercetak dilayar adalah 4321. Bagaimanakah jika baris counter (*y--*) kita hilangkan? Berapa kalikah pengulangan akan terjadi?

Contoh 8.12. Penggunaan *do-while* pada Java.

```
public class ContohDoWhile {
    public static void main(String[] args) {
        int z=3;
```

```

do {
    System.out.println("Java");
    z++;
} while (z < 6);
}

```

Contoh 8.12 menunjukkan bagaimana kita dapat menggunakan *do-while* untuk mengulang pencetakan kata "Java". Perhatikan dengan baik sintaks penulisan *do-while* pada contoh ini. *Do-while* juga membutuhkan inisialisasi dan *counter* agar pengulangan dapat dilakukan. Cobalah jalankan program di atas. Berapa kalikah tulisan "Java" akan tampil? Sekarang gantilah kondisi pada *while* dengan *z < 1*. Apakah kode program masih mencetak tulisan "Java"? Mengapa? Bacalah kembali Bab 7 pada bagian struktur kontrol pengulangan untuk memahami hal ini.

### 8.5.3. Menggunakan *Break* dan *Continue*

Pernyataan *break* memiliki tiga fungsi yaitu:

- Menghentikan pemilihan pada pernyataan *switch*.
- Menghentikan proses pengulangan atau keluar dari badan *loop*.
- Keluar dari blok label tertentu.

Pemakaian *break* pada pernyataan *switch* sudah kita pelajari pada struktur pemilihan. Kita akan pelajari sekarang bagaimana menggunakan *break* pada pengulangan. Perhatikan contoh berikut.

Contoh 8.13. Penggunaan *break* pada pengulangan.

```

class BreakPengulangan {
    public static void main(String[] args) {
        for (int i=0; i<10; i++) {
            if (i == 5) {
                break;
            }
            System.out.println("baris ke-" + i);
        }
        System.out.println("Ini setelah break pengulangan");
    }
}

```

Pada contoh 8.13 seharusnya pengulangan akan terjadi sebanyak 10 kali (dimulai dari 0 sampai dengan 9). Tetapi, karena ada pernyataan `if (i == 5) {break;}` maka pengulangan akan berhenti ketika nilai *i* = 5. Kemudian

alur program akan keluar dari badan dan menjalankan perintah setelah tanda akhir pengulangan.

Pernyataan *continue* digunakan untuk memaksa program untuk melanjutkan proses pengulangan. Perhatikan contoh berikut.

Contoh 8.14. Penggunaan *continue*.

```
String nama[] = {"Joni", "Riko", "Denis", "Riko"};
int hitung = 0;
for( int i=0; i<names.length; i++ ){
    if( !nama[i].equals("Riko") ){
        continue;
    }
    hitung++;
}
System.out.println("Ada " + hitung + " Riko dalam daftar");
```

Contoh 8.14 adalah potongan program untuk mencari jumlah nama Riko di dalam kumpulan nama. Pernyataan `if(!names[i].equals("Riko"))` mempunyai arti jika isi pada variabel nama bukan "Riko" maka jalankan perintah *continue*. Peletakkan pernyataan *continue* ini memaksa program untuk mengulang langsung tanpa harus menjalankan perintah di bawah *continue*. Artinya baris `hitung++` tidak akan dijalankan. Sehingga kalau kita eksekusi kode program di atas hasilnya adalah 2.

## 8.6. EXCEPTION HANDLING

### 8.6.1. Pengertian *Exception Handling*

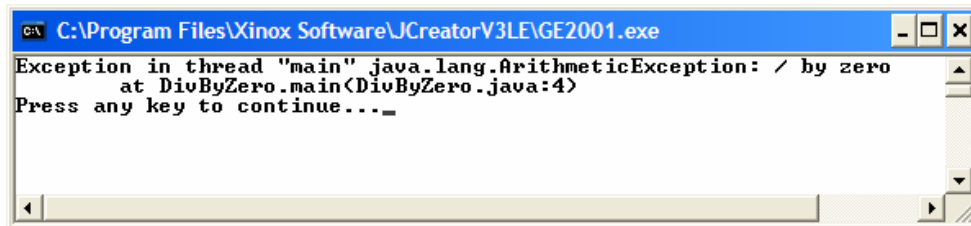
Kesalahan dalam sebuah program sering kali muncul, meskipun program tersebut dibuat oleh *programmer* berkemampuan tinggi. Untuk menghindari pemborosan waktu pada proses pencarian kesalahan, Java menyediakan mekanisme penanganan *exception*. *Exception* adalah singkatan dari *Exceptional Events*. Menurut definisi umum, *exception* adalah kondisi abnormal yang terjadi saat *runtime*. *Runtime error* atau kesalahan-kesalahan yang terjadi pada saat program berjalan diwujudkan dan *exception*. *Exception* dapat dibangkitkan secara otomatis oleh sistem Java *runtime* maupun sengaja kita buat melalui pernyataan tertentu untuk tujuan tertentu.

Perhatikan kode program berikut:

```
public class DivByZero {
    public static void main(String args[]) {
        int a = 5, b = 0, c;
        c = a/b;

        System.out.println("c = " + c);
    }
}
```

Kode program di atas, secara sintaks tidak salah. Namun ada kesalahan yang fatal yaitu ada pembagian dengan 0. Yaitu  $a = 5$  dibagi  $b = 0$ . Kesalahan seperti ini sering terjadi karena memang tidak kentara bila kita tidak cermat. Apabila kita kompilasi kode program tersebut maka kompilasi akan berlangsung sukses. Namun bila kita jalankan kita tidak akan mendapatkan hasil tetapi program akan menampilkan pesan terjadinya exception atau kondisi tidak normal (Gambar 8.8) dan program akan berhenti.



Gambar 8.8. Peringatan terjadinya kesalahan.

Pesan pada Gambar 8.8 tersebut menginformasikan tipe *exception* yang terjadi pada baris dimana *exception* itu berasal. Inilah aksi *default* yang terjadi bila terjadi *exception* yang tidak tertangani. Jika tidak terdapat kode yang menangani *exception* yang terjadi, aksi *default* akan bekerja otomatis.

Beberapa tipe *exception* yang umum antara lain:

- *ArithmeticException*. *Exception* karena kesalahan yang ada hubungannya dengan perhitungan, misalnya pembagian dengan 0.
- *ArrayIndexOutOfBoundsException*. *Exception* karena membaca indeks array diluar batas yang ditetapkan.
- *NullPointerException*. Kesalahan karena pointer yang tidak berisi (null)
- Dan lain-lain

### 8.6.2. Try dan Catch

*Try* digunakan untuk membuat blok yang berisi pernyataan-pernyataan yang mungkin menimbulkan *exception*. Apabila dalam proses eksekusi pernyataan-pernyataan pada blok tersebut terjadi *exception* maka *exception* akan dilempar ke bagian blok penangkap *exception* yang dibuat dengan kata kunci *catch*. Perhatikan contoh berikut in.

Contoh 8.15. Exception dengan try-catch.

```
public class DivByZero {
    public static void main(String args[]) {
        int a = 5, b = 0, c;

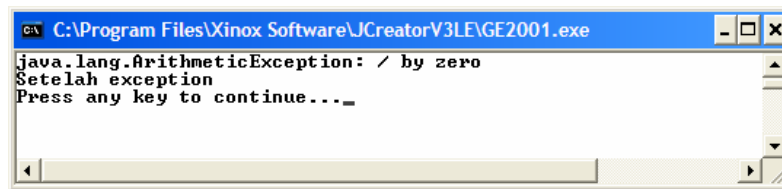
        try {
            c = a/b;
```

```

    } catch (ArithmeticException exc) {
        //Reaksi jika terjadi exception
        System.out.println(exc);
    }
    System.out.println("Setelah exception");
}
}

```

Kode program pada contoh 8.15 ini adalah pengembangan dari kode program sebelumnya. Pernyataan  $c = a/b$  merupakan pernyataan yang kita uji apakah mengandung *exception* atau tidak. Jika terjadi *exception* maka *exception* akan dilempar ke bagian *catch*. *Exception* yang kita periksa adalah *ArithmeticException*. Reaksi yang muncul jika terjadi *exception* adalah menjalankan pernyataan `System.out.println(exc);`. Dengan mekanisme seperti ini maka program tidak akan dipaksa berhenti dan perintah setelah blok *try-catch* tetap bisa dijalankan. Gambar 8.9. menunjukkan hasil eksekusi kode program di atas. Bandingkan dengan Gambar 8.8.



Gambar 8.9. Output dari try-catch.

Umumnya *exception* pada satu masalah tidak hanya satu. Pada contoh 8.16 berikut ini, kita menggunakan beberapa tipe *exception* untuk memeriksa kemungkinan terjadi *exception*. Ketik kode program berikut kemudian jalankan dan amati apa yang terjadi.

Contoh 8.16. Exception dengan try-catch.

```

class BanyakEksepsi {
    public static void test(int a, int b) {
        try {
            int c = a / b;
            System.out.println("Hasil bagi: " + c);

            int[] Arr = {1,2,3,4,5}; // array dengan 5 elemen
            Arr[10] = 11; // mengakses indeks ke-10
        } catch (ArithmeticException ae) {
            System.out.println("Terdapat pembagian dengan 0");
            System.out.println(ae);
        } catch (ArrayIndexOutOfBoundsException oobe) {
            System.out.println("Indeks di luar rentang");
            System.out.println(oobe);
        }
    }
}

```



```

public static void main(String[] args) {

    test(4, 0); // menimbulkan ArithmeticException
    System.out.println();

    test(12, 4); // menimbulkan
    ArrayIndexOutOfBoundsException
}
}

```

### 8.6.3. Throw

Disamping menangkap *exception*, Java juga mengizinkan seorang user untuk melempar (*throw*) sebuah *exception*. Perhatikan contoh 8.17 berikut ini.

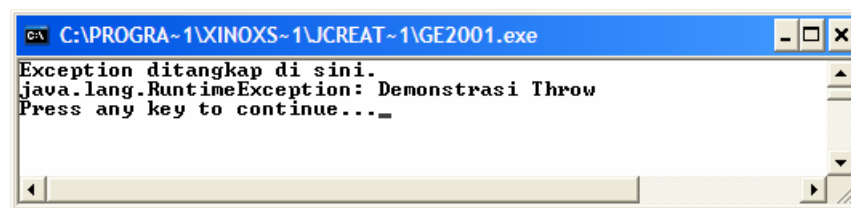
Contoh 8.17. *Exception* dengan *try-catch* dan *throw*.

```

class ThrowDemo {
    public static void main(String args[]){
        String input = "Salah input";
        try {
            if (input.equals("Salah input")) {
                throw new
RuntimeException("Demonstrasi Throw");
            } else {
                System.out.println(input);
            }
            System.out.println("Setelah throw");
        } catch (RuntimeException e) {
            System.out.println("Exception
ditangkap di sini.");
            System.out.println(e);
        }
    }
}

```

Perhatikan pada pernyataan yang dimulai dari perintah *if*. Kalau diartikan pernyataan tersebut adalah jika nilai variabel *input* sama dengan "Salah input" maka lemparkan *exception* dengan menampilkan "Demonstrasi Throw". Output dari kode program ini akan tampak seperti pada Gambar 8.10.



Gambar 8.10. Output program dengan *throw*.

#### 8.6.4. Finally

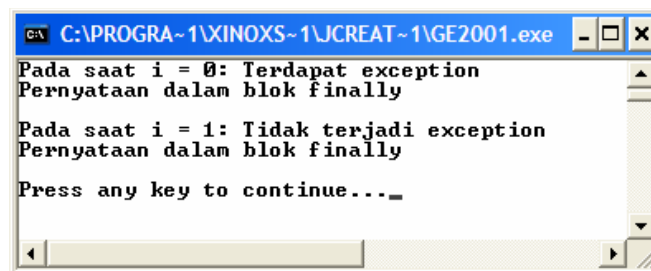
Blok *finally* mengandung kode penanganan setelah penggunaan *try* dan *catch*. Blok kode ini selalu tereksekusi apapun yang terjadi pada blok *try*. Blok kode tersebut juga akan menghasilkan nilai *true* meskipun *return*, *continue* ataupun *break* tereksekusi. Perhatikan kode program berikut.

Contoh 8.18. *Exception* dengan *try-catch-finally*.

```
class DemoFinally {
    private static int i = 0;

    public static void main(String[] args) {
        while (true) {
            try {
                System.out.print("Pada saat i = " + i + ": ");
                if (i++ == 0) {
                    throw new Exception(); // melempar exception
                }
                System.out.println("Tidak terjadi exception");
            } catch (Exception e) {
                System.out.println("Terdapat exception");
            } finally {
                System.out.println("Pernyataan dalam blok
finally\n");
                if (i == 2) {
                    break; // pada saat i==2, pengulangan akan
berhenti
                }
            }
        }
    }
}
```

Bila kode program dijalankan maka akan tampak seperti pada Gambar 8.11. Perhatikan bahwa apa yang ada pada blok kode *finally* akan selalu dijalankan.



Gambar 8.11. Output kode program *try-catch-finally*.

## 8.7. MULTI-THREADING

### 8.7.1. Pengertian Thread

Sebuah *thread* merupakan sebuah pengontrol aliran program. Untuk lebih mudahnya, bayangkanlah *thread* sebagai sebuah proses yang akan dieksekusi di dalam sebuah program tertentu. *Thread* adalah suatu bagian program yang tidak tergantung pada bagian lain dan dapat dijalankan secara bersama-sama. Hal ini berarti suatu *thread* dapat diberhentikan atau diistirahatkan tanpa harus menghentikan yang lainnya. Pada Java setiap *thread* dikontrol oleh suatu obyek unik turunan *Thread*, yang didefinisikan di dalam paket `java.lang`.

Pada saat sebuah program Java dijalankan, sebenarnya terdapat satu *thread* yang secara otomatis dibuat. *Thread* ini biasa disebut sebagai *thread* utama. *Thread* ini merupakan induk dari *thread-thread* yang lain. Meskipun *thread* utama ini otomatis dijalankan tetapi kita dapat mengendalikannya melalui obyek *Thread* dengan memanggil method `currentThread()`. Perhatikan contoh berikut.

Contoh 8.19. Thread utama.

```
class ThreadUtama {
    public static void main(String[] args)
        throws InterruptedException {

        // mendapatkan thread yang sedang aktif
        Thread tUtama = Thread.currentThread();

        // menampilkan informasi tentang thread
        System.out.print("Informasi thread: ");
        System.out.println(tUtama.toString());

        for (int i=0; i<5; i++) {
            System.out.println("Detik ke-" + (i+1));
            Thread.sleep(1000); // membuat delay selama 1 detik
        }
    }
}
```

Pada contoh di atas, kita mendefinisikan thread dengan nama `tUtama`. Variabel ini kita gunakan untuk menangkap thread utama yang sedang berjalan pada program dengan perintah `Thread.currentThread()`. Kemudian informasi tentang thread ini kita tampilkan di layar. Pada baris yang dimulai dengan `for`, kita akan menggunakan perintah untuk mengontrol thread yang sedang berjalan. Kita menggunakan *method* `sleep` untuk mengontrol *thread* agar menunda pekerjaan selama 1 detik tiap kali pengulangan. Cobalah ketik kode di atas, kemudian jalankan. Kemudian coba hapus baris `Thread.sleep(1000);`. Jalankan lagi program. Apa yang berbeda?

### 8.7.2. Pembuatan dan Penggunaan Thread

*Thread* dapat dibuat dengan dua cara yaitu dengan membuat kelas baru yang menerapkan *interface Runnable* dan membuat kelas baru dengan menurunkan dari kelas *Thread*. Kedua cara tersebut membutuhkan paket `java.lang`. Secara *default* paket ini telah otomatis diimpor pada saat kita membuat program dengan Java.

Pada bagian ini kita hanya akan membahas cara pertama. Sedangkan cara kedua akan kita pelajari secara bersamaan pada bagian *multi-thread*.

Perhatikan contoh berikut.

Contoh 8.20. Membuat thread dengan interface Runnable.

```
class TestRunnable implements Runnable {
    // mengimplementasikan method run() yang dideklarasikan
    // di dalam interface Runnable
    public void run() {
        System.out.println("Thread anak dieksekusi");
    }
}

class PenerapanRunnable {
    public static void main(String[] args) {

        // (LANGKAH KE-1): membuat objek Runnable
        TestRunnable obj = new TestRunnable();

        // (LANGKAH KE-2): membuat objek Thread dengan
        // melewati objek Runnable
        Thread t = new Thread(obj);

        // (LANGKAH KE-3) : menjalankan thread
        t.start();

        System.out.println("Thread utama dieksekusi");
    }
}
```

Pada contoh di atas kita membuat lebih dulu kelas `TestRunnable` yang menerapkan `Runnable` (perhatikan baris `class TestRunnable implements Runnable` dan blok kode dibawahnya). Kemudian kita membuat obyek `TestRunnable` dari kelas tersebut (lihat pada baris `TestRunnable obj = new TestRunnable()`). Obyek ini kita gunakan untuk membuat thread baru dengan cara menggunakan constructor kelas thread (lihat baris `Thread t = new Thread(obj)`). Setelah terbentuk maka kita dapat menjalankan thread tersebut (lihat baris `t.start()`).

### 8.7.3. Multi-Thread

Pada contoh 8.19 dan 8.20, kita hanya berhubungan dengan satu dan dua thread. Namun sesungguhnya Java member kemungkinan untuk membuat lebih dari dua thread. Kondisi ini dinamakan sebagai *Multi-thread*. Coba kalian perhatikan contoh berikut ini.

Contoh 8.21. Membuat multi-thread.

```
class MyThread1 extends Thread {
    public void run() {
        try {
            for (int i=0; i<10; i++) {
                System.out.println("Thread pertama: detik ke-" +
(i+1));
                if (i != 9) {
                    sleep(1000);
                } else {
                    System.out.println("Thread pertama
selesai...\n");
                }
            }
        } catch (InterruptedException ie) {
            System.out.println(ie.getMessage());
        }
    }
}

class MyThread2 extends Thread {
    public void run() {
        try {
            for (int i=0; i<5; i++) {
                System.out.println("Thread kedua: detik ke-" +
(i+1));
                if (i != 4) {
                    System.out.println();
                    sleep(1000);
                } else {
                    System.out.println("Thread kedua selesai...\n");
                }
            }
        } catch (InterruptedException ie) {
            System.out.println(ie.getMessage());
        }
    }
}

class DemoMultipleThread {
    public static void main(String[] args) {

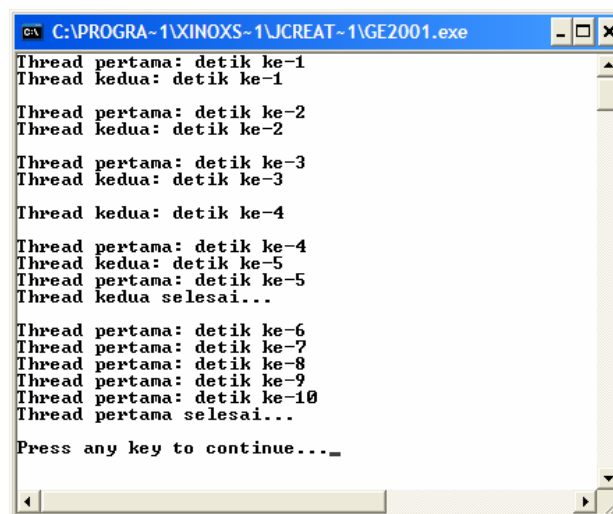
        MyThread1 t1 = new MyThread1();
        t1.start();
```

```

        MyThread2 t2 = new MyThread2();
        t2.start();
    }
}

```

Kode program di atas menunjukkan bagaimana membentuk dua buah *thread*. *Thread* yang pertama kita buat melalui pembuatan kelas **MyThread1** dengan cara menurunkan kelas **Thread**. Cara ini adalah cara kedua membuat *thread* yang kita singgung di bagian sebelumnya. *Thread* kedua dengan nama kelas **MyThread2** juga kita buat dengan cara yang sama. Kemudian pada class **DemoMultipleThread** kita membuat obyek **t1** dari kelas **MyThread1** dan obyek **t2** dari kelas **MyThread2**. Apabila dijalankan, kode program di atas akan tampak seperti pada Gambar 8.12.



Gambar 8.12. Hasil eksekusi multi-thread.

## 8.8. APLIKASI PEMROGRAMAN BERORIENTASI OBYEK DENGAN JAVA

Pada contoh-contoh sebelumnya dan beberapa penjelasannya kita telah menyinggung beberapa kali tentang *class* (kelas) dan *object* (obyek). Namun kita belum mempelajari dengan jelas apa sebenarnya kelas dan obyek dalam Java. Kelas dan obyek dalam Java adalah penerapan dari prinsip-prinsip pemrograman berorientasi obyek yang telah kita pelajari di awal bab ini.

Kelas dapat didefinisikan sebagai kerangka yang mendefinisikan variabel-variabel, method umum dari sebuah obyek tertentu. Pada pemrograman berorientasi obyek, kelas tidak jauh berbeda dengan tipe data primitive. Perbedaannya, tipe data digunakan untuk mendeklarasikan variabel normal, sedangkan kelas digunakan untuk mendeklarasikan variabel yang berupa obyek. Kelas masih bersifat abstrak.

### 8.8.1. Pembuatan Kelas

Pada Java kelas didefinisikan dengan kata kunci `class`. Bentuk umum untuk mendefinisikan kelas adalah sebagai berikut:

```
class NamaKelas
{
    tipe data1;
    tipe data2;
    ...
    tipe dataN;

    tipe method1 (daftar parameter) {
        //blok kode untuk method1
    }
    tipe method2 (daftar parameter) {
        //blok kode untuk method2
    }
    ...
    tipe methodN (daftar parameter) {
        //blok kode untuk methodN
    }
}
```

Data atau variabel yang didefinisikan di dalam kelas sering disebut sebagai *instance variable*. Nilai-nilai variabel ini akan dibaca melalui *method-method* yang tersedia. Dengan demikian method digunakan sebagai antarmuka (interface) antara pemakai kelas dengan data yang ada di dalam kelas tersebut. Ingat kembali prinsip *encapsulation* di awal bab. Perhatikan contoh kelas berikut.

Contoh 8.22. Membuat kelas sederhana.

```
class Siswa
{
    String name;
    String alamat;
    int usia;
}
```

Pada kode di atas kita membuat class dengan nama Siswa. Ada tiga data yang ada pada class tersebut yaitu nama, alamat dan usia. Kita belum menambahkan method di sini. Melalui kode di atas sebenarnya kita telah mendefinisikan tipe data baru yaitu Siswa. Kode program di atas hanyalah sebuah template. Artinya kalau jalankan program di atas tidak akan menghasilkan apa-apa. Kita perlu membuat obyek aktual berdasarkan kelas di atas. Dengan cara sebagai berikut.

Contoh 8.23. Menggunakan kelas.

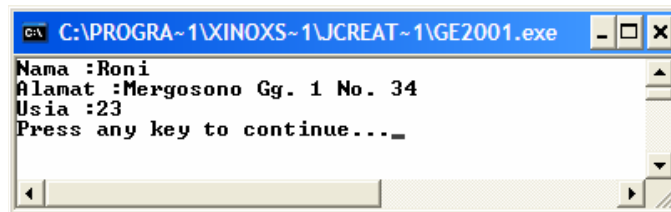
```
class Siswa {
    String nama;
    String alamat;
    int usia;
}
```

```

public class DataSiswa {
    public static void main(String[] args) {
        Siswa siswal = new Siswa();
        siswal.nama = "Roni";
        siswal.alamat = "Mergosono Gg. 1 No. 34";
        siswal.usia = 23;
        System.out.println("Nama :" + siswal.nama);
        System.out.println("Alamat :" + siswal.alamat);
        System.out.println("Usia :" + siswal.usia);
    }
}

```

Kode program di atas harus kita simpan dengan nama file DataSiswa.java, buka Siswa.java. Hal ini karena *method* main berada pada class DataSiswa. Pada kode di atas kelas Siswa kita gunakan pada kelas DataSiswa. Kita membuat obyek aktual dari kelas Siswa dengan cara menetikkan **Siswa siswal = new Siswa();**. Siswa1 adalah nama obyek aktual dari kelas Siswa. Setelah itu baru kita dapat menggunakan variabel atau data yang ada pada kelas siswa. Kalau dijalankan, kode program di atas akan menghasilkan output seperti pada Gambar 8.13.



Gambar 8.13. Hasil eksekusi terhadap class DataSiswa.

Sekarang kita akan buat kelas menjadi sedikit lebih kompleks dengan mengikutkan method pada kelas tersebut. Perhatikan contoh berikut.

Contoh 8.23. Pembuatan kelas yang mempunyai method.

```

class Siswa {
    String nama;
    String alamat;
    int usia;
    double nilaiMatematika;
    double nilaiBhsInggris;
    double nilaiBhsIndonesia;
    double rerata;

    // Menghasilkan nama dari Siswa
    public String getName(){
        return nama;
    }
}

```



```

        // Mengubah nama siswa
        public void setName( String temp ){
            nama = temp;
        }

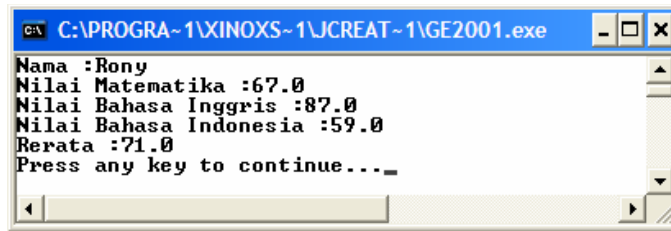
        // Menghitung rata - rata nilai
        public double getRerata(){
            rerata = (
nilaiMatematika+nilaiBhsInggris+nilaiBhsIndonesia )/3;
            return rerata;
        }
    }

    public class DataSiswa {
        public static void main(String[] args) {
            Siswa siswal = new Siswa();
            siswal.setName("Rony");
            siswal.nilaiMatematika = 67;
            siswal.nilaiBhsInggris = 87;
            siswal.nilaiBhsIndonesia = 59;

            System.out.println("Nama          : "          +
siswal.getName());
            System.out.println("Nilai Matematika : " +
siswal.nilaiMatematika);
            System.out.println("Nilai Bahasa Inggris : " +
siswal.nilaiBhsInggris);
            System.out.println("Nilai Bahasa Indonesia : " +
siswal.nilaiBhsIndonesia);
            System.out.println("Rerata : " +
siswal.getRerata());
        }
    }

```

Pada kode di atas kita memperluas kelas Siswa dengan menambahkan empat variabel yaitu nilaiMatematika, nilaiBhsInggris, nilaiBhsIndonesia dan rerata. Kita juga menambahkan tiga buah *method* yaitu getName, setName dan getRerata. getName merupakan method untuk menampilkan isi dari variabel nama. setName adalah *method* untuk memberi nilai pada variabel nama. getRerata adalah *method* untuk menghitung rata-rata nilai dari tiga pelajaran dan menampilkan isi dari hasil perhitungan. Perhatikan bagaimana method ini digunakan pada kelas DataSiswa. Apabila dijalankan maka kita akan memperoleh output seperti pada Gambar 8.14.



Gambar 8.14. Eksekusi pada class yang mempunyai method.

Ada beberapa tipe *method* di dalam class, yaitu *method* yang tidak mengembalikan nilai, *method* yang mengembalikan nilai dan *method* khusus yaitu constructor. Secara umum *method* ini boleh kita samakan dengan prosedur atau fungsi (lihat kembali Bab 6 dan 7). Perhatikan contoh-contoh berikut ini.

Contoh 8.24. Pembuatan method tanpa pengembalian nilai.

```

Class Bangun {
    double panjang;
    double lebar;

    // Mendefinisikan method void (tidak mengembalikan nilai)
    void cetakLuas() {
        System.out.println("Luas bangun = " +
            (panjang * lebar));
    }
}

class pakaiBangun {
    public static void main(String[] args) {
        Bangun b1, b2;

        // instansiasi objek
        b1 = new Bangun();
        b2 = new Bangun();

        // mengisi data untuk objek b1
        b1.panjang = 4;
        b1.lebar = 3;

        // mengisi data untuk objek b2
        b2.panjang = 6;
        b2.lebar = 5;

        // memanggil method cetakLuas() untuk masing-masing
        objek
        b1.cetakLuas();
        b2.cetakLuas();
    }
}

```

```
}
```

Pada kode di atas, class Bangun mempunyai satu *method* yaitu cetakLuas. *Method* ini tidak mengembalikan nilai. Hasil akhir dari *method* ini akan tersimpan pada *method* tersebut. Bandingkan dengan contoh berikut.

Contoh 8.25. Pembuatan *method* dengan pengembalian nilai.

```
Class Bangun {
    double panjang;
    double lebar;

    // Mendefinisikan method yang mengembalikan nilai
    double hitungLuas() {
        double luas = panjang * lebar;
        return luas;
    }
}

class pakaiBangun {
    public static void main(String[] args) {
        Bangun b1, b2;

        // instansiasi objek
        b1 = new Bangun();
        b2 = new Bangun();

        // mengisi data untuk objek b1
        b1.panjang = 4;
        b1.lebar = 3;

        // mengisi data untuk objek b2
        b2.panjang = 6;
        b2.lebar = 5;

        // memanggil method hitungLuas() untuk masing-masing
        objek
        System.out.println("Luas b1 = " + b1.hitungLuas());
        System.out.println("Luas b2 = " + b2.hitungLuas());
    }
}
```

Pada contoh 8.25 ini kita membuat *metode* hitungLuas yang mengembalikan nilai. Perhatikan deklarasi *method* tidak lagi menggunakan *void* tetapi menggunakan *double* yang merupakan tipe data nilai yang dikembalikan. Untuk mengembalikan nilai digunakan kata kunci *return*. Perhatikan cara pemanggilan *method* dari contoh 8.24 dan 8.25, apa yang berbeda?

*Method* dapat juga memiliki argumen seperti halnya pada fungsi atau prosedur. Perhatikan contoh berikut.

Contoh 8.26. Pembuatan method dengan argumen.

```
class Bangun {
    double panjang;
    double lebar;

    // method dengan argumen
    void isiData(double p, double l) {
        panjang = p;
        lebar = l;
    }

    // method yang mengembalikan nilai
    double hitungLuas() {
        double luas = panjang * lebar;
        return luas;
    }
}

class pakaiBangun {
    public static void main(String[] args) {
        Bangun b;

        // instansiasi obyek
        b = new Bangun();

        // memanggil method isiData dan mengisi argumennya
        b.isiData(6,8);

        // memanggil method hitungLuas() untuk objek b
        System.out.println("Luas b = " + b.hitungLuas());
    }
}
```

Pada contoh 8.26 ini kita menambahkan satu method lagi pada class Bangun yaitu isiData. Karena tidak mengembalikan nilai maka kita menggunakan void. Method ini mempunyai dua buah argument yaitu p dan l yang digunakan untuk menampung nilai yang akan kita masukkan. Perhatikan bagaimana kita menggunakan method ini (lihat bagian **b.isiData(6, 8)** ).

*Constructor* yang merupakan method khusus, merupakan *method* yang didefinisikan di dalam kela dan akan dipanggil secara otomatis setiap kali terjadi pendefinisian (instansiasi) obyek. Biasanya *constructor* berfungsi untuk melakukan inisialisasi nilai terhadap data-data yang terdapat pada kelas yang bersangkutan. Nama method constructor harus sama dengan nama kelas itu sendiri. *Constructor* tidak memiliki kembalian nilai dan tidak void. Perhatikan contoh berikut ini.

Contoh 8.27. Pembuatan class dengan constructor.

```

class Bangun {
    double panjang;
    double lebar;

    // constructor dengan argumen
    Bangun(double p, double l) {
        panjang = p;
        lebar = l;
    }

    // method yang mengembalikan nilai
    double hitungLuas() {
        double luas = panjang * lebar;
        return luas;
    }
}

class pakaiBangun {
    public static void main(String[] args) {
        Bangun b;

        // instansiasi obyek
        b = new Bangun();

        // memanggil method isiData dan mengisi argumennya
        b.isiData(6,8);

        // memanggil method hitungLuas() untuk objek b
        System.out.println("Luas b = " + b.hitungLuas());
    }
}

```

Kode di atas sekilas sama dengan contoh 8.26, namun sebenarnya berbeda. Pada kode ini terdapat *constructor* dengan nama yang sama dengan kelasnya yaitu Bangun. *Constructor* seperti halnya method yang lain boleh mempunyai argumen dan boleh tidak. Pada contoh di atas kita *constructor* Bangun mempunyai argumen p dan l.

### 8.8.2. Penerapan Inheritance

Prinsip inheritance atau pewarisan, secara umum telah kalian pelajari di awal bab. Pewarisan adalah keuntungan besar dalam pemrograman beorientasi obyek karena suatu sifat atau method yang telah didefinisikan dalam superclass, secara otomatis diwariskan dari pada semua subclasses. Jadi kita mungkin hanya perlu mendefinisikan method satu kali di superclass kemudian dapat kita gunakan pada subclass. Perhatikan contoh berikut.

Contoh 8.28. Penerapan inheritance.

```

class SuperA {
    private int a;

    public void setSuperA(int nilai) {
        a = nilai;
    }
    public int getSuperA() {
        return a;
    }
}

// membuat kelas turunan (subclass) dari kelas A
class SubB extends SuperA {
    private int b;

    public void setSubB(int nilai) {
        b = nilai;
    }
    public int getSubB() {
        return b;
    }
}

class DemoKelasTurunan1 {
    public static void main(String[] args) {

        // melakukan instansiasi terhadap kelas B
        SubB ObyekB = new SubB();

        // mengeset nilai objek dari kelas B
        ObyekB.setSuperA(50);
        ObyekB.setSubB(200);

        // mendapatkan nilai yang terdapat dalam objek dari
        kelas B
        System.out.println("Nilai a : " + Obyek.getSuperA());
        System.out.println("Nilai b : " + Obyek.getSubB());
    }
}

```

Pada kode di atas, class SuperA adalah super class yang memiliki satu data yaitu a dan dua method yaitu setSuperA dan getSuperA. Class SubB adalah turunan dari kelas SuperA (perhatikan deklarasi **class SubB extends SuperA**). Data dan method yang ada pada class SuperA secara otomatis akan dibawa ke class SubB. Sehingga class SubB akan mempunyai dua data yaitu a dan b. Data a adalah hasil warisan dari class SuperA sedang data b adalah milik class SubB sendiri. Method pada class SubB akan terdiri dari empat method, yaitu setSuperA dan getSuperA yang merupakan warisan dari class SuperA serta class setSubB dan getSubB yang milik class SubB sendiri. Cobalah ketik program di atas kemudian jalankan dan cermati hasil yang diperoleh.

Perhatikan contoh yang lain berikut ini.

Contoh 8.29. Penerapan inheritance untuk menghitung luas dan volume.

```
class Bangun {
    protected double panjang;
    protected double lebar;

    // constructor default
    Bangun() {
        panjang = lebar = 0;
    }

    Bangun(double p, double l) {
        panjang = p;
        lebar = l;
    }

    // method yang mengembalikan nilai
    public double hitungLuas() {
        double luas = panjang * lebar;
        return luas;
    }
}

class Box extends Bangun {
    private double tinggi;

    // constructor class Box
    Box (int p, int l, int t) {
        panjang = p;
        lebar = l;
        tinggi = t;
    }

    public double getTinggi() {
        return tinggi;
    }

    public double hitungVolume() {
        double volume = panjang * lebar * tinggi;
        return volume;
    }
}

class inheritBangun {
    public static void main(String[] args) {
        Box kotak;

        // instansiasi obyek
        kotak = new Box(6, 8, 3);
    }
}
```

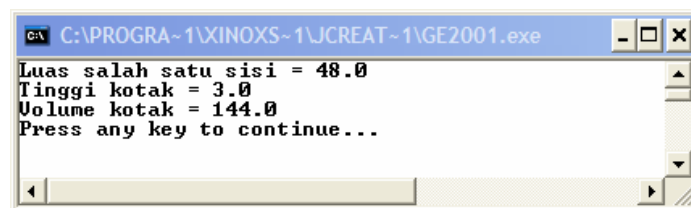
```

        // memanggil method hitungLuas(), getTinggi() dan
        hitung volume()
        System.out.println("Luas salah satu sisi = " +
        kotak.hitungLuas());
        System.out.println("Tinggi kotak = " +
        kotak.getTinggi());
        System.out.println("Volume kotak = " +
        kotak.hitungVolume());
    }
}

```

Kelas Bangun di atas adalah superclass sedangkan Box adalah subclass. Pada default constructor Bangun, nilai panjang dan lebar diinisialisasi dengan nilai 0. Perhatikan didepan deklarasi variabel panjang dan lebar pada kelas Bangun dicantumkan kata kunci `protected` yang berarti kelas turunan dari Bangun tetap dapat mengakses nilai dari variabel tersebut namun yang tidak memiliki hubungan turunan tidak dapat mengaksesnya.

Pada Box yang merupakan subclass ditambahkan satu variabel yaitu tinggi dengan tambahan kata kunci `private` diikuti tipe datanya. `Private` menunjukkan bahwa variabel tinggi hanya bisa diakses didalam kelas Box. Lawan dari `private` adalah `public`, yang berarti dapat diakses oleh siapapun dan dari manapun. Pada kelas Box juga kita tambahkan satu method yaitu `hitungVolume()`. Pada contoh 8.29 di atas obyek yang kita buat yaitu kotak, merupakan hasil instansiasi dari kelas Box. Oleh karena kelas Box adalah turunan dari kelas Bangun, maka kita dapat mengakses method `hitungLuas()` yang merupakan warisan dari kelas Bangun. Tentu saja kita bisa mengakses method `getTinggi()` dan `hitungVolume()` yang merupakan method pada kelas Box. Kalau kita eksekusi program di atas, maka akan tampak seperti berikut.



Gambar 8.15. Hasil eksekusi program kelas Bangun dan Box.

### 8.8.3. Penerapan Overriding dan Overloading

Kadang-kadang, ketika kita membuat method di dalam subclass, kita ingin membuat method dengan nama yang sama dengan method pada superclass namun dengan penerapan yang berbeda. Sebagai contoh pada kelas Bangun pada contoh 8.29, tersedia method `hitungLuas()`. Misalnya kita ingin membuat subclass Segitiga yang merupakan turunan dari kelas Bangun. Kemudian kita ingin membuat method `hitungLuas()` yang penerapannya tidak lagi  $\text{luas} = \text{panjang} \times \text{lebar}$  tetapi dengan penerapan  $\text{luas} = 0.5 \times \text{alas} \times \text{tinggi}$ . Pada kondisi ini method `hitungLuas()` dari superclass akan tertutupi oleh method pada



subclass. Hal seperti ini biasa disebut sebagai *overriding*. Perhatikan contoh berikut.

Contoh 8.30. Penerapan overriding.

```
class Bangun {
    // method umum
    public double hitungLuas() {
        System.out.println("Method belum terdefinisi");
        Return 0;
    }
}

class Segitiga extends Bangun {

    private double alas;
    private double tinggi;

    Segitiga (int a, int t) {
        alas = a;
        tinggi = t;
    }

    // overriding method hitungLuas()
    public double hitungLuas() {
        double luas = 0.5 * alas * tinggi;
        return luas;
    }
}

class overridingBangun {
    public static void main(String[] args) {
        Segitiga s;

        // instansiasi obyek
        s = new Segitiga(6, 8);

        // memanggil method hitungLuas() dari subclass Segitiga
        System.out.println("Luas segitiga = " +
s.hitungLuas());
    }
}
```

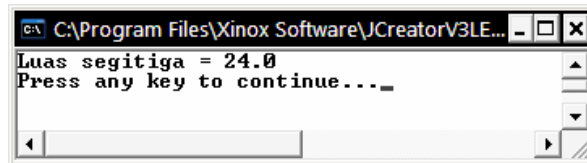
Pada contoh di atas, kelas Bangun sebagai superclass mempunyai method `hitungLuas()`, tetapi belum kita definisikan isi methodnya. Pada kelas Segitiga, method `hitungLuas()` ini kita overriding untuk mendapatkan nilai luas segitiga. Pada eksekusi program di atas, yang dijalankan adalah method `hitungLuas()` yang ada pada *subclass* Segitiga. Hasil dari eksekusi program akan tampak seperti pada Gambar 8.16. Apabila kita ingin tetap menjalankan method `hitungLuas()` yang ada pada kelas Bangun kita dapat memanggil dengan kata kunci *super*. Rubahlah method `hitungLuas()` pada kelas Segitiga dengan kode berikut.

```

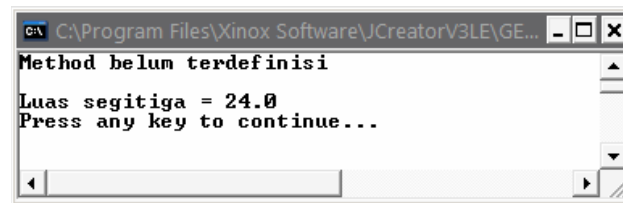
public double hitungLuas() {
    super.hitungLuas();
    System.out.println();
    double luas = 0.5 * alas * tinggi;
    return luas;
}

```

Jalankan program, maka kalian akan mendapatkan hasil seperti Gambar 8.17. Bandingkan dengan hasil eksekusi sebelumnya (Gambar 8.16)



Gambar 8.16. Hasil eksekusi overriding pada method hitungLuas().



Gambar 8.17. Hasil eksekusi overriding dan pernyataan super.

Overloading memiliki kesamaan dengan overriding dalam hal menyembunyikan method dari superclass. Tetapi memiliki perbedaan yaitu, pada overloading methodnya memiliki nama yang sama dengan method pada kelas induk, tetapi memiliki daftar argument dan implementasi yang berbeda. Sedangkan overriding, method mempunyai nama dan daftar argumen yang sama dengan kelas induk dan hanya implementasinya yang berbeda. Perhatikan contoh berikut.

Contoh 8.31. Contoh overloading.

```

class Bangun {
    // method umum
    public double hitungLuas() {
        System.out.println("Method belum terdefinisi");
        return 0;
    }
}

```

```

class BujurSangkar extends Bangun {
    private double sisi;

    // overload method hitungLuas()
    public double hitungLuas(double sisi) {
        double luas = sisi * sisi;
        return luas;
    }
}

class overloadBangun {
    public static void main(String[] args) {
        BujurSangkar b;

        // instansiasi obyek
        b = new BujurSangkar();

        // memanggil method hitungLuas() dari subclass
        BujurSangkar
        System.out.println("Luas BujurSangkar = " +
        b.hitungLuas(6));
    }
}

```

Perhatikan pada kode di atas, method `hitungLuas()` pada kelas `Bangun` tidak mempunyai argumen sedangkan pada kelas `BujurSangkar` mempunyai argumen yaitu `sisi`. Inilah yang disebut sebagai *overloading*. Bandingkan dengan contoh *overriding* pada contoh 8.29 dan 8.30.

#### 8.8.4. Penerapan Polymorphism

Seperti telah dijelaskan di awal bab, *polymorphism* adalah kemampuan sebuah untuk menerjemahkan suatu method menjadi berbagai macam aksi. Sebenarnya ketika kalian mempelajari *overriding* dan *overloading* di atas secara tidak langsung kalian telah mempelajari dasar-dasar penyusunan teori *polymorphism*. *Polymorphism* membolehkan kelas induk (*superclass*) untuk mendefinisikan method umum untuk semua turunannya. Kelas-kelas turunannya dapat mengimplementasikan method tersebut sesuai dengan karakteristik masing-masing kelas.

Pada contoh 8.30 dan 8.31 kita telah membuat kelas `Bangun` yang merupakan kelas induk yang mempunyai method `hitungLuas()`. Method ini bersifat umum. Pada contoh 8.30, method ini kita implementasikan dengan untuk mencari nilai luas segitiga pada kelas `Segitiga`. Sedangkan pada contoh 8.31, method ini kita implementasikan untuk mencari luas `BujurSangkar` pada kelas `BujurSangkar`. Perhatikan contoh berikut ini.

Contoh 8.32. Penerapan polymorphism.

```
class Bangun {
    public double hitungLuas() {
        System.out.println("Method umum");
        return 0;
    }
}

class BujurSangkar extends Bangun {
    private double sisi;

    BujurSangkar(int s) {
        sisi = s;
    }

    //overriding method hitungLuas()
    public double hitungLuas() {
        double luas = sisi * sisi;
        return luas;
    }
}

class Segitiga extends Bangun {
    private double alas;
    private double tinggi;

    Segitiga (int a, int t) {
        alas = a;
        tinggi = t;
    }

    // overriding method hitungLuas()
    public double hitungLuas() {
        double luas = 0.5 * alas * tinggi;
        return luas;
    }
}

class Lingkaran extends Bangun {
    private double jarijari;
    private final double PI = 3.1416;

    Lingkaran(int r) {
        jarijari = r;
    }

    //overriding method hitungLuas()
    public double hitungLuas() {
        double luas = PI * jarijari * jarijari;
        return luas;
    }
}
```

```

}

class DemoPolimorfisme2 {
    public static void main(String[] args) {

        Bentuk obyek;
        BujurSangkar b = new BujurSangkar(12);
        Segitiga s = new Segitiga(5, 6);
        Lingkaran l = new Lingkaran(4);

        // obyek mengacu pada objek BujurSangkar
        obyek = b;
        // akan memanggil method yang terdapat pada
        BujurSangkar
        System.out.println("Luas bujursangkar : " +
        obyek.hitungLuas());
        System.out.println();

        // obyek mengacu pada objek Segitiga
        obyek = s;
        // akan memanggil method yang terdapat pada Segitiga
        System.out.println("Luas segitiga : " +
        obyek.hitungLuas());
        System.out.println();

        // obyek mengacu pada objek Lingkaran
        obyek = l;
        // akan memanggil method yang terdapat pada Lingkaran
        System.out.println("Luas lingkaran: " +
        obyek.hitungLuas());
        System.out.println();
    }
}

```

Pada contoh 8.32 ini kita menggabungkan contoh-contoh sebelumnya untuk menunjukkan bagaimana *polymorphism* terbentuk. Kelas induk adalah Bangun dan mempunyai *subclass* yaitu BujurSangkar, Segitiga dan Lingkaran. Semuanya *subclass* mempunyai *method* hitungLuas() yang diturunkan dari kelas Bangun. Perhatikan bahwa meskipun nama method hitungLuas() ada pada semua *subclass*, ternyata penerapannya berbeda-beda tergantung pada *subclass* masing-masing.

#### 8.8.5. Menggunakan Paket dan Interface

Pada beberapa bagian di atas telah disinggung tentang *package* atau paket. *Packages* dalam JAVA berarti pengelompokan beberapa class dan interface dalam satu unit. Fitur ini menyediakan cara untuk mengatur *class* dan *interface* dalam jumlah banyak dan menghindari kekacauan pada penamaan *class* dan *file*.

Kalau kalian cermati contoh 8.4 pada sub bab terdahulu, sebenarnya kalian telah menggunakan konsep *package* ini secara tidak langsung. Pada contoh tersebut kita menggunakan pernyataan `import` untuk menggunakan kelas-kelas yang ada pada paket `Java.IO`. Java menyediakan banyak paket yang telah dibuat oleh tim pengembang java. Paket-paket ini dapat kita gunakan untuk mempermudah pemrograman

Cara membuat paket cukup mudah, kita tinggal menggunakan kata kunci `package` diikuti dengan nama paket yang kita inginkan seperti berikut:

**`package NamaPaket;`**

`NamaPaket` adalah nama paket yang kita akan gunakan untuk menyimpan file kode program dan file hasil kompilasi. Java menggunakan sistem direktori untuk menyimpan paket yang dibuat. Apabila kita membuat paket dengan nama `PaketBangun`, maka kita harus membuat direktori dengan nama yang sama persis yaitu `PaketBangun`. Dan file paket dan kelas-kelas dalam satu paket tersebut harus kita simpan pada direktori tersebut. Perhatikan contoh berikut.

#### Contoh 8.33. Pembuatan paket.

Pada bagian ini kita akan membuat paket yang merupakan modifikasi dari contoh 8.32. Nama paket yang ingin kita buat adalah `PaketBangun` yang mempunyai tiga anggota kelas yaitu kelas `BujurSangkar`, `Segitiga` dan `Lingkaran`. Untuk tahap awal, buatlah direktori dengan nama `PaketBangun` (pada contoh ini, direktori ini kita letakkan di `D:\TestCode\Java\PaketBangun`). Kemudian secara berturut-turut buatlah kode program berikut dan simpan dengan nama sesuai nama kelasnya pada direktori `PaketBangun`. Perhatikan pada awal kode kelas selalu diawali pernyataan `package PaketBangun;`. Hal ini menunjukkan bahwa kelas tersebut merupakan anggota dari `PaketBangun`.

##### File : `BujurSangkar.java`

```
package PaketBangun;
class BujurSangkar extends Bangun {
    private double sisi;

    public BujurSangkar(int s) {
        sisi = s;
    }
    public double hitungLuas() {
        double luas = sisi * sisi;
        return luas;
    }
}
```

##### File : `Segitiga.java`

```
package PaketBangun;
class Segitiga extends Bangun {
    private double alas;
    private double tinggi;

    public Segitiga (int a, int t) {
```

```

        alas = a;
        tinggi = t;
    }
    public double hitungLuas() {
        double luas = 0.5 * alas * tinggi;
        return luas;
    }
}

```

File : Lingkaran.java

```

package PaketBangun;
class Lingkaran extends Bangun {
    private double jarijari;
    private final double PI = 3.1416;

    public Lingkaran(int r) {
        jarijari = r;
    }
    public double hitungLuas() {
        double luas = PI * jarijari * jarijari;
        return luas;
    }
}

```

Setelah itu lakukan pengaturan classpath seperti terlihat pada sub bab awal. Dan lakukan kompilasi pada ketiga file tersebut di atas sehingga akan diperoleh hasil file seperti Gambar 8.18.

```

C:\
D:\TestCode\Java>cd PaketBangun
D:\TestCode\Java\PaketBangun>javac BujurSangkar.java
D:\TestCode\Java\PaketBangun>javac Segitiga.java
D:\TestCode\Java\PaketBangun>javac Lingkaran.java
D:\TestCode\Java\PaketBangun>dir /w
Volume in drive D is Data
Volume Serial Number is 7A1D-08A5

Directory of D:\TestCode\Java\PaketBangun

[.]                [..]                BujurSangkar.class
BujurSangkar.java   Lingkaran.class   Lingkaran.java
[PaketBangun]      Segitiga.class   Segitiga.java
                   6 File(s)        1,814 bytes
                   3 Dir(s)        21,579,771,904 bytes free

D:\TestCode\Java\PaketBangun>

```

Gambar 8.18. Kompilasi pada tiga file anggota paket.

Setelah kompilasi berhasil, buatlah file baru di direktori D:\TestCode\Java, misalnya dengan nama PakaiPaketBangun.java. Kemudian ketikkan kode berikut ini pada file tersebut. Kompilasi dan jalankan kode program tersebut.

```
//import seluruh kelas pada PaketBangun
import PaketBangun.*;

class PakaiPaketBangun {
    public static void main(String[] args) {

        BujurSangkar b = new BujurSangkar(12);
        Segitiga s = new Segitiga(5, 6);
        Lingkaran l = new Lingkaran(4);

        System.out.println("Luas bujursangkar : " +
b.hitungLuas());
        System.out.println();

        System.out.println("Luas segitiga : " +
s.hitungLuas());
        System.out.println();

        System.out.println("Luas lingkaran: " +
l.hitungLuas());
        System.out.println();
    }
}
```

Perhatikan cara penulisan untuk memanggil paket. Jika kita hanya membutuhkan kelas `Lingkaran` saja, maka perlu menuliskan `import PaketBangun.Lingkaran;`. tetapi jika kita butuh semua kelas maka kita menuliskan dengan cara `import PaketBangun.*;`. Jika kita jalankan perintah ini maka hasilnya akan sama persis dengan contoh 8.33. Perbedaannya adalah pada contoh 8.32 kelas-kelas berada pada satu file. Sedangkan pada contoh 8.33 kelas-kelas berada pada filenya masing-masing dan baru dikelompokkan dengan menggunakan *package*.

*Interface* atau antar muka pada bahasa pemrograman Java sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah *interface* dapat diimplementasikan oleh kelas lain. Sebuah kelas dapat mengimplementasikan lebih dari satu *interface*. Metode pada *interface* yang diimplementasikan pada suatu kelas harus sama persis dengan yang ada pada *interface*. *Interface* digunakan jika kita ingin kelas yang tidak berhubungan mengimplementasikan method yang sama. Perhatikan contoh berikut ini.

Contoh 8.34. Pembuatan interface.

```
interface Bentuk {
    public double luas();
    public double volume();
}
```



Pada contoh ini kita membuat sebuah interface dengan nama Bentuk yang mempunyai dua method yaitu luas() dan volume(). Perhatikan penulisan interface. Kedua method yang ada dideklarasikan tanpa isi. Kita dapat menggunakan interface ini untuk membuat kelas baru dan mengimplementasikan interface ini dalam kelas tersebut. Perhatikan contoh berikut ini.

Contoh 8.34. Penggunaan interface.

```
class Kubus implements Bentuk {

    int x = 10;
    public double luas( ) {
        return (6 * x * x);
    }

    public double volume() {
        return (x * x * x);
    }

}

class Lingkaran implements Bentuk {

    int radius = 10;
    public double luas() {
        return (Math.PI * radius * radius);
    }
    public double volume() {
        return 0;
    }

}
```

## 8.9. RINGKASAN

- Konsep penting harus pahami dalam pemrograman berorientasi yaitu, kelas, obyek, abstraksi, enkapsulasi, *inheritance* dan *polymorphism*.
- Ada 8 tipe data dasar pada Java yaitu *boolean* (untuk bentuk logika), *char* (untuk bentuk tekstual), *byte*, *short*, *int*, *long* (*integral*), *double* and *float* (*floating point*).
- Operator yang tersedia pada Java adalah operator aritmatika, relasional, logika dan bitwise.
- Struktur kontrol program untuk percabangan dapat dilakukan dengan pernyataan *if*, *if ... else*, dan *if ...else ... if*, sedangkan pengulangan dapat dilakukan dengan *for*, *while* dan *do-while*.
- *Exception* adalah kondisi abnormal yang terjadi saat *runtime*. Java menyediakan mekanisme *try*, *throw*, *catch* dan *finally* untuk menangani exception.

- *Thread* adalah suatu bagian program yang tidak tergantung pada bagian lain dan dapat dijalankan secara bersama-sama. Java membolehkan kita untuk mengatur thread sesuai dengan kebutuhan.
- Java mendukung penuh konsep kelas, inheritance, overriding, overloading, dan polymorphism.
- Paket adalah kumpulan dari kelas-kelas. Sedangkan interface adalah kelas tanpa atribut dan mempunyai method yang dideklarasikan tanpa isi.

#### 8.10. SOAL-SOAL LATIHAN

1. Buatlah program untuk menampilkan nilai dari beberapa variabel yang mempunyai tipe data dan nilai awal sebagai berikut:
  - a. Tipe data float, nilai awal = 3.45.
  - b. Tipe data char, nilai awal = B
  - c. Tipe data int, nilai awal = 23
2. Buatlah program untuk menghitung nilai rata-rata dari tiga variabel berikut : number1 = 56.3, number2 = 12 dan number3 = 2.78.
3. Dengan menggunakan variabel pada no 2, buatlah program untuk mencari nilai terkecil dari ketiga variabel tersebut.
4. Ambil tiga nilai ujian dari user dan hitung nilai rata-rata dari nilai tersebut. Berikan pernyataan "Selamat" pada output jika nilai rata-rata lebih besar atau sama dengan 60, selain itu beri output. Gunakan `BufferedReader` untuk mendapat input dari user, dan `System.out` untuk output hasilnya.
5. Buat sebuah program yang mencetak nama kalian selama seratus kali. Buat tiga versi program ini menggunakan while loop, do while dan for-loop.
6. Dengan menggunakan kelas, buatlah program tentang kelas staf sekolah. Kelas ini adalah superclass yang mempunyai subclass guru dan tenaga administrasi.
  - Kelas staf sekolah mempunyai variabel nama, alamat, jumlah anak, tanggal awal bekerja dan pangkat. Selain itu juga punya method lama bekerja (diperoleh dari tanggal saat ini dikurangi tanggal awal bekerja) dan method hitung gaji pokok. Gaji pokok diperoleh dari nilai gaji dasar ditambah tunjangan lama, tunjangan pangkat bekerja dan tunjangan anak. Selain itu mempunyai method hitung gaji total yang belum didefinisikan.
  - Kelas guru, selain variabel dan method pada superclassnya, juga mempunyai variabel bidang keahlian dan method hitung tunjangan mengajar yang diperoleh dari banyaknya jam mengajar dikali upah per jam mengajar. Definiskan method hitung gaji total sebagai penjumlahan gaji pokok dengan tunjangan mengajar.

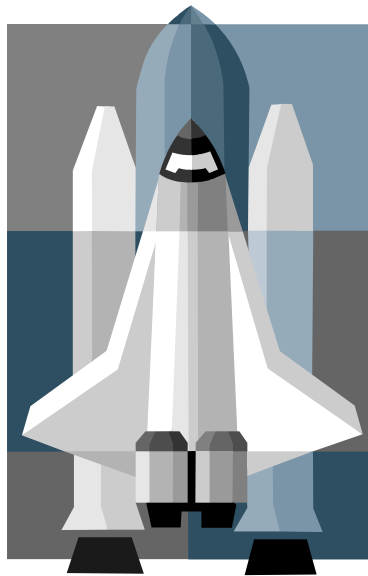
- Kelas tenaga administrasi, selain variabel dan method pada superclassnya, juga mempunyai variabel jam kerja seminggu dan method hitung upah lembur. Upah lembur dihitung jika seorang tenaga administrasi bekerja lebih dari 40 jam perminggu. Kelebihannya adalah jam lembur. Upah lembur = jumlah jam lembur dikali dengan upah per jam lembur. Definisikan method hitung gaji total sebagai penjumlahan gaji pokok dengan upah lembur.



---

**BAB 9 PEMROGRAMAN APLIKASI DENGAN C++**

---



(Sumber: Clip Art Microsoft Office 2007)

Gambar 9.1. Pesawat Luar Angkasa.

Gambar 9.1 di samping ini mungkin menimbulkan pertanyaan. Apa hubungannya dengan judul bab di atas? Tapi tahukah anda bahwa pesawat luar angkasa seperti pada Gambar 9.1 dikendalikan oleh seperangkat komputer canggih yang sebagian aplikasi pengendalinya ditulis dengan bahasa pemrograman C++.

Standar kompetensi program aplikasi menggunakan C++ terdiri atas lima kompetensi dasar. Dalam penyajian pada buku ini, setiap kompetensi dasar memuat tujuan, uraian materi, dan latihan. Rangkuman diletakkan pada akhir bab. Kompetensi dasar pada bab ini adalah dasar-dasar pemrograman C++, menerapkan fungsi, menerapkan pointer, menerapkan konsep Class, dan merancang aplikasi berorientasi Obyek. Sebelum mempelajari

kompetensi ini ingatlah kembali sistem operasi, algoritma pemrograman dasar, algoritma pemrograman lanjutan dan konsep-konsep pemrograman berorientasi obyek pada bab-bab sebelumnya.

Pada bagian akhir, tercantum soal-soal latihan yang disusun dari soal-soal yang mudah hingga soal-soal yang sulit. Latihan soal ini digunakan untuk mengukur kemampuan terhadap kompetensi dasar yang ada pada bab ini. Untuk melancarkan kemampuan agar lebih baik dalam mengerjakan soal, disarankan semua soal dalam latihan ini dapat dikerjakan di sekolah dengan bimbingan guru maupun di rumah.

## TUJUAN

Setelah mempelajari bab ini diharapkan pembaca akan mampu :

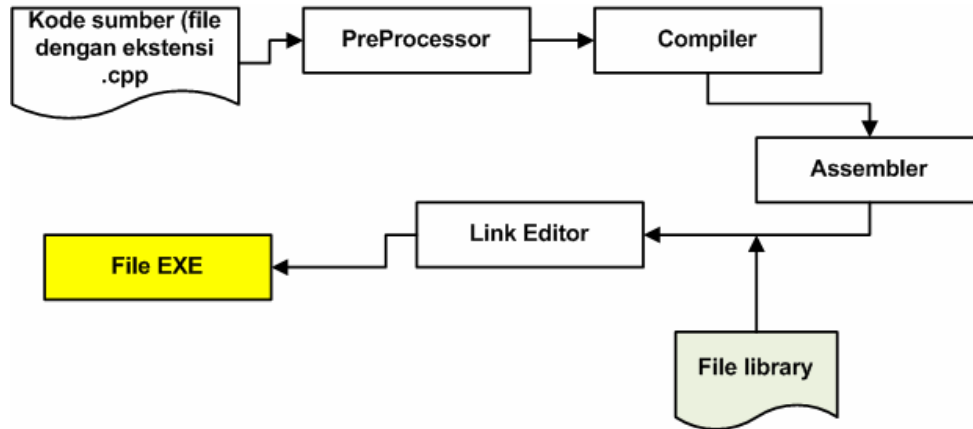
- o Menjelaskan dasar-dasar pemrograman C++
- o Menerapkan fungsi
- o Menerapkan pointer
- o Menerapkan konsep kelas
- o Merancang aplikasi berorientasi obyek

### 9.1. DASAR-DASAR PEMROGRAMAN C++

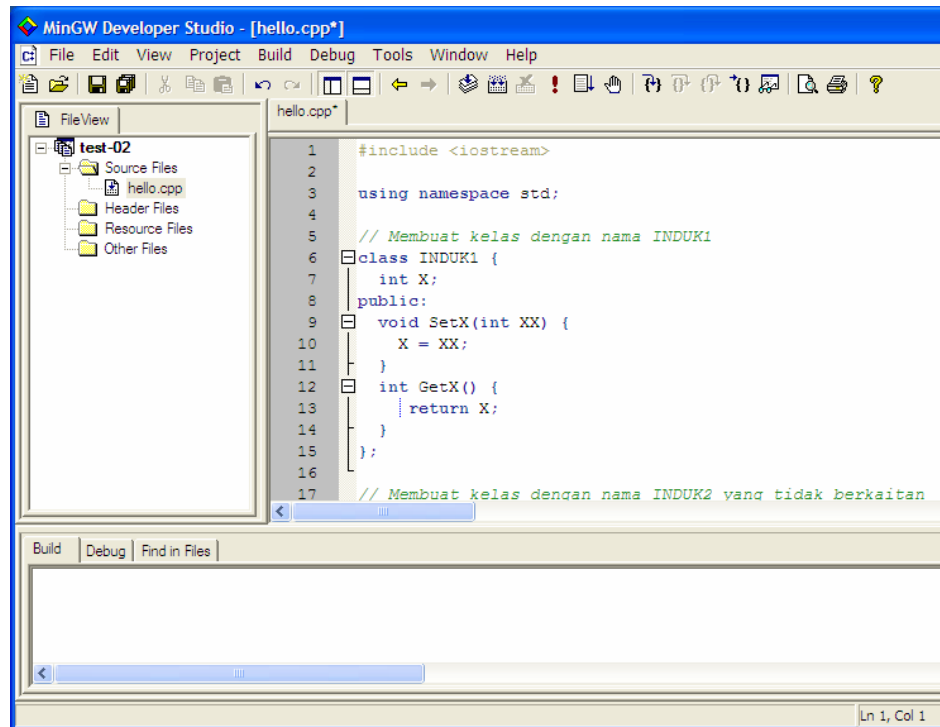
Bahasa C++ merupakan bahasa yang sangat populer di dunia pengembangan perangkat lunak. Seperti halnya pendahulunya yaitu Bahasa C, C++ juga dimasukkan dalam kelompok bahasa tingkat menengah (middle level language). Tujuan utama pembuatan C++ adalah untuk meningkatkan produktivitas pemrogram dalam membuat aplikasi. Keistimewaan C++ adalah karena bahasa ini telah mendukung OOP (Object Oriented Programming). Selain itu juga tersedia banyak pustaka (library) yang dapat kita gunakan untuk mempercepat pembuatan aplikasi. Pustaka ini sebagian tersedia gratis di beberapa situs internet.

Kode program dalam bahasa C++ yang kita buat tidak dapat langsung dieksekusi namun harus dikompilasi lebih dahulu dengan *compiler* C++ yang sesuai. Konsep kompilasi dan eksekusi program pada C++ dapat digambarkan seperti pada Gambar 9.2. Kode program yang kita buat disebut sebagai kode sumber dan merupakan file teks biasa dengan nama file yang berekstensi .cpp. Kode program ini kemudian dimasukkan ke *PreProcessor*. Keluaran dari *PreProcessor* ini adalah file yang akan dimasukkan ke dalam *Compiler*. *Compiler* akan menerjemahkan kode program dalam file tersebut menjadi bahasa *assembly*. Kode program program ini kemudian diproses oleh *Assembler* menjadi kode obyek. Jika tidak ada file pustaka (*library*) yang terlibat, maka kode obyek ini akan langsung dieksekusi menjadi file. Jika ada pustaka lain yang terlibat maka *Link Editor* akan mengkombinasikan kode obyek dan pustaka untuk membentuk file EXE.

Kita dapat mengetikkan kode-kode C++ dengan sebarang teks editor, seperti Notepad, Vi, atau yang lainnya. Namun akan lebih mudah jika kita menggunakan perangkat IDE (Integrated Development Environment) yang menyediakan secara terintegrasi teks editor dan compiler C++. Beberapa IDE yang cukup terkenal antara lain Microsoft Visual Studio, Borland C++, MingGW Developer Studio, dan lain-lain. Pada buku ini sebagian besar code ditulis dan dijalankan dengan menggunakan MingGW Developer Studio (Gambar 9.3). Lisensi IDE ini free, artinya kita bisa menggunakan tanpa diributkan dengan masalah lisensi dari perangkat lunak. Kalian dapat mendownload IDE ini dari alamat internet <http://www.parinya.ca/>.



Gambar 9.2. Proses kompilasi pada C++



Gambar 9.3. MingGW Developer Studio

### 9.1.1. Kerangka Program C++

```
#include <iostream>

using namespace std;

void nama_fungsi() {
    //kode untuk nama_fungsi
    .....;
}

// Fungsi utama
int main() {

    // kode bagian main/utama
    .....;

    return 0;
}
```

Perhatikan kode program di samping ini. Kode program terdiri dari beberapa bagian seperti berikut:

- Bagian untuk mendaftarkan file atau kondisi tertentu. Bagian ini selalu diawali dengan tanda #.
- Bagian pendefinisian fungsi. Diawali dengan kata kunci `void` diikuti nama fungsi. Bagian ini boleh tidak ada jika kita tidak membuat fungsi.
- Bagian `main()`. Pada kode program biasanya dimulai dengan `int main()`. Bagian ini harus ada pada setiap program karena merupakan fungsi utama.
- Bagian yang dimulai dengan tanda `{` dan diakhiri dengan tanda `}`, baik pada bagian `void` maupun `main`, disebut blok kode.

### 9.1.2. Header File (.h)

File header yaitu file dengan ekstensi `.h` merupakan file yang berisi fungsi-fungsi dan telah dikompilasi sebelumnya. File ini biasanya berisi fungsi-fungsi tertentu yang nantinya akan kita gunakan dalam kode program yang kita buat. Sebagai contoh file `iostream.h` mempunyai sejumlah fungsi untuk menampilkan output dan menangkap input seperti `cout` dan `cin`.

Ada dua model penulisan file header, yaitu dengan dituliskan lengkap dengan ekstensinya dan tidak. Bentuk pertama adalah model penulisan untuk compiler C++ lama (Contoh 9.1). Bentuk kedua merupakan bentuk yang didukung oleh Compiler C++ standar (Contoh 9.2).

Contoh 9.1. Penulisan file header pada Compiler C++ lama.

```
#include <iostream.h>
Int main() {
    ...
    return 0;
}
```

Contoh 9.2. Penulisan file header pada Compiler C++ standar.

```
#include <iostream>
using namespace std;
Int main() {
    ...
}
```



```

        return 0;
    }

```

### 9.1.3. Komentar, Identifier dan Tipe Data

- **Komentar**

Komentar pada C++ dapat dilakukan dengan dua cara, yaitu pertama, dengan tanda // dan diletakkan sebelum komentar dan kedua dengan tanda /\* yang ditutup dengan \*/. Cara pertama digunakan jika komentar hanya terdiri dari satu baris, sedangkan cara kedua jika komentar lebih dari satu baris.

Contoh 9.3. Komentar dengan tanda //.

```

// program pertama
#include <iostream>
int main( ){
    cout << "Hello World"; // cetak "Hello World" di layar
}

```

Contoh 9.4. Komentar dengan tanda /\* .. \*/.

```

/* Program pertama
Ditulis oleh ARM
Tanggal 17-11-2007 */
#include <iostream>
int main( ){
    std::cout << "Hello World"; // cetak "Hello World" di
    layar
}

```

- **Identifier**

Identifier atau nama dari variabel atau konstanta pada C++ secara umum sama mengikuti aturan umum penamaan yang telah dijelaskan pada Bab 5. Ada satu tambahan yang cukup penting dalam pembuatan identifier pada C++ yaitu, bersifat case sensitif. Variabel dengan nama **namaguru** berbeda dengan variabel **NamaGuru**.

Ada dua cara untuk mendeklarasikan konstanta. Pertama, dengan menggunakan preprocessor directive #define. Kedua, dengan menggunakan kata kunci const. Ketiklah program berikut ini, kemudian periksalah hasil dari eksekusinya.

Contoh 9.5. Pendeklarasian konstanta.

```

#include <iostream>
#define potongan 0.1;
using namespace std;

int main() {
    const float hargaPerUnit = 2500;
    int jumlahUnit;
}

```

```

float hargaTotal, hargaDiskon, diskon;
cout << "Masukkan jumlah unit pembelian : ";
cin >> jumlahUnit;
hargaTotal = jumlahUnit * hargaPerUnit;
diskon = hargaTotal * potongan;
hargaDiskon = hargaTotal - diskon;
cout << "Total harga pembelian = " << hargaTotal <<
endl;
cout << "Diskon = " << diskon << endl;
cout << "Harga Diskon = " << hargaDiskon;
return 0;
}

```

Pada contoh di atas, terdapat dua konstanta yaitu potongan dan hargaPerUnit yang masing-masing dideklarasikan dengan cara berbeda.

Deklarasi variabel dilakukan dengan cara menyebutkan lebih dahulu tipe datanya kemudian diikuti dengan nama variabelnya seperti pada contoh 9.5. Pada contoh tersebut ada beberapa variabel yang telah didefinisikan lebih dahulu yaitu jumlahUnit, hargaTotal, hargaDiskon dan diskon. Masing-masing dengan tipe datanya sendiri. Untuk beberapa variabel yang tipe datanya sama, kita dapat menggabungkan dalam satu baris dengan tanda pemisah koma. Perhatikan pada contoh di atas. hargaTotal, hargaDiskon dan diskon sama-sama mempunyai tipe data float sehingga penulisannya digabung.

Kalian harus mendeklarasikan dengan lengkap seluruh variabel atau konstanta yang akan dipakai lebih dahulu sebelum menggunakan. Kalau tidak maka program tidak akan dapat dieksekusi.

## • Tipe data

Seperti telah dijelaskan pada Bab 5, tipe data tergantung dari yang disediakan oleh bahasa pemrograman. Pada C++ tipe data dasar yang disediakan dapat dilihat pada tabel berikut. Kalau kalian perhatikan tipe-tipe ini sama persis dengan Java. Hal ini karena Java sebenarnya banyak mengambil elemen-elemen bahasa pemrograman dari C++.

Tabel 9.1. Tipe data pada C++.

Tipe Data	Keterangan
int	Tipe data bilangan bulat dengan ukuran 4 bytes
long	Tipe data bilangan bulat namun lebih besar dari int.
float	Tipe data bilangan pecahan
double	Tipe data bilangan pecahan namun lebih besar dari float
char	Tipe data karakter yang berisi huruf, angka atau simbol-simbol (alphanumeric) sepanjang berada pada tanda " " atau ' '.
bool	Tipe data boolean
short	Tipe data bilangan bulat dengan ukuran 2 bytes

Tipe data bentukan yang tersedia pada C++ adalah array, struct dan enum. Array akan kita bahas pada bagian lain di bab ini. Sedangkan struct dan enum telah kita singgung di Bab 5. Namun begitu cara penulisannya belum kita ulas. Cobalah ketikkan kode-kode program pada contoh berikut ini kemudian eksekusi untuk melihat hasilnya.

Contoh 9.6. Penggunaan tipe data struct.

```
#include <iostream>
using namespace std;

int main() {

    struct Guru {
        char* NIP;
        char* Nama;
        char* Alamat;
    };

    Guru A;

    A.NIP = "132 232 477";
    A.Nama = "Syafiq";
    A.Alatmat = "Perum. Dirgantara Permai";

    // Menampilkan nilai yang diisikan ke layar
    cout<<A.NIP<<endl;
    cout<<A.Nama<<endl;
    cout<<A.Alatmat<<endl;

    return 0;
}
```

Contoh 9.6. Penggunaan tipe data enum.

```
#include <iostream>
using namespace std;

enum JENIS_KELAMIN { Pria, Wanita };
int main() {

    struct Guru {
        char* NIP;
        char* Nama;
        JENIS_KELAMIN JK;
    } A;

    A.NIP = "132 232 477";
    A.Nama = "Syafiq";
    A.JK = Pria;

    cout<<"NIP      : "<<A.NIP<<endl;
    cout<<"Nama      : "<<A.Nama<<endl;
```

```

        cout<<"Jenis Kelamin      : "<<A.JK<<endl;

    return 0;
}

```

#### 9.1.4. Operator

Seperti halnya pada VB dan Java yang telah anda pelajari, C++ juga menyediakan banyak operator yang dapat kita gunakan untuk membantu memecahkan masalah tertentu. Secara umum banyak kemiripan antara Java dengan C++ dalam penyediaan operator. Operator-operator yang tersedia pada C++ dapat dilihat pada Tabel 9.2.

Tabel 9.2. Operator pada C++.

Jenis Operator	Fungsi	Contoh
<b>Operator assignment</b>		
=	Memasukkan (assign) nilai ke dalam suatu variabel	C = 5
<b>Operator unary</b>		
+	Membuat nilai positif	X = +10
-	Membuat nilai negatif	Y = -12
++	Menaikkan nilai variabel satu nilai	++C (pre-increment) C++ (post-increment)
--	Menurunkan nilai variabel satu nilai	--C (pre-increment) C-- (post-increment)
<b>Operator Binary</b>		
+	Penjumlahan	3 + 5 = 8
-	Pengurangan	7 - 2 = 5
*	Perkalian	5 * 2 = 10
/	Pembagian	6 / 3 = 2
%	Sisa hasil bagi (modulus)	5 / 2 = 1
<b>Operator Logika</b>		
&&	AND	1 && 1 = 1
	OR	1    0 = 1
!	NOT	!0 = 1
<b>Operator Relasional</b>		
>	Lebih besar	(5 > 4) = 1
<	Lebih kecil	(5 < 4) = 0
>=	Lebih besar atau sama dengan	(4 >= 4) = 1
<=	Lebih kecil atau sama dengan	(5 <= 4) = 0
==	Sama dengan	(5 ==4) = 0
!=	Tidak sama dengan	(5 != 4) = 1

Operator Bitwise		
&	AND	$1 \& 0 = 0$
	OR	$1   0 = 1$
^	XOR	$1 \wedge 1 = 0$
~	NOT	$\sim 0 = 1$
Operator Ternary		
?:	Digunakan jika melibatkan tiga operand	

### 9.1.5. Struktur Kontrol Program

Struktur kontrol program pada C++ secara umum sama dengan VB dan Java yang telah kalian pelajari. Yang berbeda adalah sintaks penulisannya. Untuk percabangan, C++ menyediakan perintah *if* (tanpa *then*) dan *switch ... case*. Sedangkan untuk pengulangan, C++ menyediakan perintah *for*, *while*, dan *do-while*. Selain itu, pada pengulangan juga menyediakan fasilitas *break* dan *continue*. Secara berurutan berikut ini akan disampaikan kode-kode program untuk menerapkan struktur kendali. Ketikkan kode-kode tersebut, kemudian jalankan dan amati apa yang terjadi.

Contoh 9.7. Penggunaan struktur percabangan *if* sederhana.

```
#include <iostream>
using namespace std;

int main() {
    int bil;

    cout<<"Masukkan sebuah bilangan bulat : ";
    cin>>bil;

    // Melakukan pengecekan bilangan dengan operator modulus
    if (bil % 2 == 0) {
        cout<<bil<<" adalah bilangan genap" << endl;
    } else {
        cout<<bil<<" adalah bilangan ganjil" << endl;
    }

    return 0;
}
```

Pada contoh di atas kita menggunakan *if* untuk memeriksa apakah suatu bilangan genap atau ganjil. Operator yang kita gunakan adalah *modulus* (%). Perhatikan cara penulisan struktur *if* dan *else*.

Contoh 9.8. Penggunaan struktur percabangan *if* tiga kondisi.

```
#include <iostream>
using namespace std;
```

```

int main() {
    int bil;

    cout<<"Masukkan sebuah bilangan bulat : ";
    cin>>bil;

    if (bil > 0) {
        cout<<bil<<" adalah bilangan POSITIF";
    } else if (bil < 0) {
        cout<<bil<<" adalah bilangan NEGATIF";
    } else {
        cout<<"ini bilangan NOL";
    }

    return 0;
}

```

Contoh 9.8. merupakan pengembangan dari Contoh 9.7. Struktur pemilihan dikembangkan menjadi tiga kondisi dengan menambahkan *else*. Apabila pemilihan lebih dari tiga kondisi atau banyak, C++ menyediakan perintah *switch ... case* untuk memudahkan proses pemilihan. Perhatikan contoh berikut.

Contoh 9.9. Penggunaan struktur percabangan dengan *switch ... case*.

```

#include <iostream>
using namespace std;

int main() {
    int bil;

    cout<<"Masukkan sebuah bilangan (1 s/d 5) : ";
    cin>>bil;

    switch (bil) {
        case 1 : cout<<"Bilangan anda adalah : SATU";
                break;
        case 2 : cout<<"Bilangan anda adalah : DUA";
                break;
        case 3 : cout<<"Bilangan anda adalah : TIGA";
                break;
        case 4 : cout<<"Bilangan anda adalah : EMPAT";
                break;
        case 5 : cout<<"Bilangan anda adalah : LIMA";
                break;
        default : cout<<"Anda memasukkan di luar batas";
    }

    return 0;
}

```

Contoh 9.9. adalah contoh percabangan dengan *switch* untuk konversi dari bilangan angka ke teks. Perhatikan bagaimana penulisan *switch* dan *case*.

Seperti juga pada Java, pengulangan dengan *for* digunakan jika kita mengetahui dengan pasti berapa banyak pengulangan akan dilakukan. Berikut contoh penggunaan pengulangan dengan *for*.

Contoh 9.10. Penggunaan struktur pengulangan dengan *for*.

```
#include <iostream>
using namespace std;

int main() {
    int C, J;
    cout<<"Cetak angka dari kecil ke besar :"<<endl;
    for (C=0; C<10; C+3) {
        cout<<C+1<<endl;
    }

    cout<<endl;
    cout<<"Cetak angka dari besar ke kecil"<<endl;
    for (J=10; J>0; J--) {
        cout<<J<<endl;
    }

    return 0;
}
```

Contoh 9.10. merupakan contoh penggunaan *for* untuk kasus sederhana. Ada dua pengulangan *for* di atas, yaitu mencetak angka dari 1 sampai dengan 10 dan dari 10 sampai dengan 1. Perhatikan penggunaan increment ++ dan --. Cara penulisan di atas sangat mirip dengan penulisan pada Java yang telah kalian pelajari sebelumnya. Pada contoh 9.11 berikut ini, pengulangan *for* dikembangkan menjadi sedikit lebih rumit dengan menerapkan *for* bersarang.

Contoh 9.11. Penggunaan struktur pengulangan dengan *for* bersarang.

```
#include <iostream>
using namespace std;

int main() {
    for (int j=1; j<=4; j++) {
        for (int k=1; k<=3; k++) {
            cout<<k+j<<' ';
        }
        cout<<endl;
    }

    return 0;
}
```

Ada dua pengulangan pada contoh 9.11. Yang pertama dengan menggunakan *j* sebagai variabel *counter*. Sedangkan yang kedua dengan variabel *counter* *k* yang bersarang di dalam pengulangan *j*. Menurut kalian, bagaimanakah output dari kode program di atas?

Penggunaan while pada pengulangan, tidak berbeda jauh dengan apa yang telah kalian pelajari pada Java maupun VB. Perhatikan contoh 9.12 dan 9.13 berikut ini.

Contoh 9.12. Penggunaan struktur pengulangan dengan while sederhana.

```
#include <iostream>
using namespace std;

int main() {
    int C;
    C = 1;           // inisialisasi nilai C

    while (C<10) {
        cout<<"Saya tidak nakal lagi"<<endl;
        C++;         // increment
    }

    return 0;
}
```

Contoh 9.13. Penggunaan struktur pengulangan dengan while bersarang.

```
#include <iostream>
using namespace std;

int main() {
    int x, y;
    x = 1;           //inisialisasi variabel x

    while (x<=4){
        y = 1;       //inisialisasi variabel y
        while (y<=3){
            cout<<y+x<<' ';
            y++;
        }
        cout<<endl;
        x++;
    }

    return 0;
}
```

Perhatikan contoh 9.12 di atas, bagaimanakah menurut kalian output dari kode program tersebut. Coba bandingkan dengan contoh 9.11. Cobalah memindahkan inisialisasi untuk variabel y. Letakkan setelah inisialisasi variabel x. Bagaimanakah hasilnya?

Bentuk berikutnya dari pengulangan pada C++ adalah dengan menggunakan do-while. Agak sedikit berbeda dengan while, kondisi pemeriksaan pada do-while diletakkan di akhir badan loop. Contoh 9.14 memberikan gambaran bagaimana do-while digunakan dalam C++.



Contoh 9.14. Penggunaan struktur pengulangan dengan do-while.

```
#include <iostream>
using namespace std;

int main() {
    int J = 5;
    int K;

    do {
        K = 1;
        do {
            cout<<K*J<<' ';
            K++;
        } while (K <= J);
        cout<<endl;
        J--;
    } while (J >= 1);

    return 0;
}
```

Contoh 9.14 terlihat menampilkan kode program yang cukup rumit. Namun bila kita cermati, ini adalah kode program dengan pengulangan do while yang bersarang. Perhatikan cara penulisan dan alur logika programnya. Do-while yang luar menggunakan variabel counter J dan ini adalah pengulangan dari besar ke kecil (perhatikan J diinisialisasi dengan nilai 10 dan syarat kondisi pada while J >=1). Sedangkan do-while yang dalam variabel counter nya adalah K dengan pengulangan dari kecil ke besar. Bagaimanakah hasil dari eksekusi kode program di atas? Perhatikan output di bawah ini. Cobalah untuk menelusuri kode program sehingga kalian benar-benar mengerti mengapa output program bisa menjadi seperti ini.

```
5 10 15 20 25
4 8 12 16
3 6 9
2 4
1
```

#### 9.1.6. Input/Output

Sampai dengan bagian ini, kalian telah cukup banyak latihan kode pemrograman C++. Namun kita belum sekalipun mempelajari statement input / output pada C++. Sebenarnya kalau kalian jeli, kalian telah secara tidak langsung mempelajari pernyataan input dan output. Perintah *cout* dan *cin* yang telah kalian gunakan adalah pernyataan input / output yang paling sering digunakan.

Baik perintah *cout* dan *cin* digolongkan sebagai *stream* yang termasuk dalam kelas *iostream*. Karena itulah setiap kali kita ingin menggunakan dua perintah tersebut kita harus memanggil file *header iostream* di awal program.

*Stream* adalah peralatan logika yang berguna untuk mendapatkan atau memberikan informasi. *Stream* berhubungan dengan perangkat keras seperti *keyboard*, layar monitor, printer melalui sistem I/O.

Perintah *cin* merupakan *stream* untuk input standar. Perintah ini akan merekam apa yang kita ketikkan dari *keyboard*. Perhatikan cara penulisannya pada contoh 9.15.

Contoh 9.15. Penggunaan *cin* dan *cout*.

```
#include <iostream>
using namespace std;

int main() {
    int bil1, bil2;

    //cin bagian satu
    cout<<"Masukkan bilangan pertama : ";
    cin>>bil1;
    cout<<"Masukkan bilangan kedua : ";
    cin>>bil2;
    cout<<"Hasil kali kedua bilangan = "<<bil1*bil2<<endl;

    //cin bagian dua
    cout<<"Masukkan dua buah bilangan : ";
    cin>>bil1>>bil2;
    cout<<"Hasil kali kedua bilangan = "<<bil1*bil2<<endl;

    return 0;
}
```

Perintah *cin* dapat digunakan untuk memasukkan data satu per satu seperti pada contoh 9.15 (lihat bagian di bawah *//cin bagian satu*) atau memasukkan data secara langsung berurutan (lihat bagian di bawah *//cin bagian dua*). Perintah *cin* harus diikuti operator *>>*.

Perintah *cout* adalah perintah melakukan output standar yaitu pada layar monitor. Perintah *cout* harus diikuti operator *<<*. Perhatikan contoh 9.15 di atas. *cout* dapat digunakan untuk mencetak langsung karakter (ditandai dengan *"* dan diakhiri dengan *"*) atau isi variabel. Seperti halnya *cin*, *cout* dapat digunakan untuk output satu persatu atau berurutan sekaligus. Pada contoh di atas pernyataan *endl* adalah pernyataan untuk mencetak baris baru.

## 9.2. FUNGSI DALAM C++

Fungsi dalam C++ memegang peranan sangat penting. Hal ini karena sebenarnya program dalam C++ adalah kumpulan dari fungsi-fungsi. Fungsi-fungsi yang telah tersedia maupun yang kita buat sendiri ini akan dipanggil dalam fungsi utama yaitu *main()*. Seperti semua bahasa pemrograman, C++ menyediakan *built-in function* yang dapat diakses dengan lebih dulu memanggil file header-nya di awal program yang kita buat. C++ juga menyediakan fasilitas

untuk membuat fungsi sendiri (*user-defined function*). Pada sub bab ini kita akan mempelajari bagaimana membuat fungsi sendiri.

### 9.2.1. Tipe-tipe Fungsi

Ada dua jenis fungsi, yaitu fungsi yang tidak mengembalikan nilai dan fungsi yang mengembalikan nilai.

- **Fungsi yang tidak mengembalikan nilai**

Fungsi ini dibuat dengan tipe *void*. Dalam VB atau Pascal, fungsi ini dikenal sebagai prosedur. Perhatikan contoh 9.16 berikut.

Contoh 9.16. Fungsi tanpa pengembalian nilai.

```
#include <iostream>
using namespace std;

// Membuat fungsi cetak angka
void CetakAngka() {
    for (int C=0; C<10; C++) {
        cout<<C+1<<endl;
    }
}

// Fungsi utama dalam program C++
int main() {

    // Memanggil fungsi CetakAngka
    CetakAngka();

    return 0;
}
```

Pada contoh ini kita membuat fungsi dengan nama *CetakAngka* dengan tipe *void* sehingga tidak mengembalikan nilai (*return value*). Perhatikan bagaimana mendeklarasikan fungsi. Fungsi akan berjalan sampai akhir kode pada fungsi tersebut.

- **Fungsi yang mengembalikan nilai**

Fungsi ini akan mengembalikan sebuah nilai untuk digunakan pada bagian program yang lain. Untuk mendefinisikan fungsi tipe ini, kita tidak menggunakan *void*, tetapi langsung tipe data dari nilai yang akan dikembalikan oleh fungsi tersebut. Perhatikan contoh 9.17 berikut.

Contoh 9.17. Fungsi dengan pengembalian nilai.

```
#include <iostream>
```

```

using namespace std;

// Membuat fungsi dengan nilai pengembalian tipe char
char* NilaiChar() {
    return "Ini nilai yang akan dikembalikan";
}

// Fungsi utama
int main() {
    // Memanggil dan menampilkan hasil fungsi
    cout<< NilaiChar();

    return 0;
}

```

Pada tipe fungsi ini, kita membutuhkan pernyataan *return* untuk menunjukkan bagian yang akan dikembalikan nilainya. Pada contoh di atas, tipe data dari nilai yang akan dikembalikan adalah *char*. Bentuk *char* dengan tanda \* menunjukkan variabel **NilaiChar** boleh berisi lebih dari satu huruf dan akan disimpan/dicetak sebagaimana ketika kita memasukkan isinya.

### 9.2.2. Penggunaan Parameter dalam Fungsi

Seperti halnya pada VB dan Java, fungsi pada C++ juga membolehkan digunakan parameter atau argumen untuk melewatkan input atau menampung output dari fungsi tersebut. Perhatikan contoh-contoh berikut.

Contoh 9.18. Fungsi dengan parameter input.

```

#include <iostream>
using namespace std;

// Membuat fungsi dengan parameter input
int Kuadrat(int X) {
    int hasil;
    hasil = X * X;
    return hasil;
}

int main() {
    int Bil, HASIL;
    cout<<"Masukkan sebuah bilangan bulat : ";
    cin>>Bil;
    HASIL = Kuadrat(Bil);          //memanggil fungsi kuadrat
    cout<<"Kuadrat dari bilangan "<<Bil<<" adalah :
    "<<HASIL;

    return 0;
}

```

Fungsi pada contoh 9.18 adalah Kuadrat. Fungsi ini membutuhkan satu variabel input (pada contoh dinamakan X). Pada fungsi main(), variabel Bil

adalah variabel yang digunakan untuk menyimpan nilai yang digunakan sebagai parameter pada fungsi Kuadrat ketika dipanggil.

Contoh 9.19. Fungsi dengan parameter input dan output.

```
#include <iostream>
using namespace std;

// Membuat fungsi dengan parameter input
int Kuadrat(int X, int *hasil) {
    *hasil = X * X;
    return *hasil;
}

int main() {
    int Bil, HASIL;
    cout<<"Masukkan sebuah bilangan bulat : ";
    cin>>Bil;
    // Menampilkan nilai setelah diproses di dalam fungsi
    cout<<"Kuadrat dari bilangan "<<Bil<<" adalah :
"<<Kuadrat(Bil, &HASIL);

    return 0;
}
```

Contoh 9.19 merupakan pengembangan dari Contoh 9.18. Pada fungsi Kuadrat kita tambahkan parameter output yaitu hasil. Parameter keluaran, harus dilewatkan berdasarkan alamat memorinya (yaitu hasil), sehingga harus menggunakan pointer (lihat tanda \* sebelum variabel hasil. Demikian juga cara pemanggilan fungsinya, parameter input dan outputnya harus disebutkan. Parameter output yang menyimpan hasil perhitungan harus kita beri awalan &.

### 9.3. POINTER DAN ARRAY

Bahasa C++ membolehkan kita untuk memanipulasi memori dengan cara penggunaan *pointer*. Hal ini merupakan fitur yang tidak disediakan oleh bahasa pemrograman yang lain. Apabila digunakan secara benar maka akan sangat menguntungkan, tetapi bila salah dalam penggunaan bisa berakibat pada kerusakan (*crash* atau *hang*) pada sistem operasi.

#### 9.3.1. Konsep dan Pengertian Pointer

*Pointer* adalah variabel. Namun berbeda dengan variabel normal, *pointer* menyimpan alamat pada memori, bukan nilai yang kita masukkan. Perhatikan contoh berikut.

Contoh 9.20. Mendeklarasikan pointer.

```
#include <iostream>

using namespace std;
```

```

int main() {
    long *Alamat;
    long X;

    Alamat = &X;
    X = 5;      // Mengisikan nilai 5 ke dalam variabel X

    cout<<"Nilai X : "<<X<<endl;
    cout<<"Nilai *Alamat : "<<*Alamat<<endl;
    cout<<"Nilai Alamat : "<<Alamat<<endl;
    cout<<"Nilai &X : "<<&X<<endl;

    *Alamat = 20;  // Mengisikan nilai 20 ke dalam *Alamat

    cout<<"Nilai X : "<<X<<endl;
    cout<<"Nilai *Alamat : "<<*Alamat<<endl;
    cout<<"Nilai Alamat : "<<Alamat<<endl;
    cout<<"Nilai &X : "<<&X<<endl;

    return 0;
}

```

Pada contoh di atas, kita mendeklarasikan variabel alamat sebagai *pointer* dengan menambahkan tanda \* di depan nama variabel. Jika kita tidak menggunakan tanda \* berarti variabel tersebut akan berfungsi seperti variabel normal. Variabel X kita deklarasikan sebagai variabel normal dengan tipe data long. Perhatikan pada baris `Alamat = &X;`. Baris ini menyatakan bahwa variabel Alamat (bukan pointer) akan diisi dengan nilai dari alamat dari X. Tanda & di depan nama variabel berarti kita menginginkan nilai alamat memorinya yang kita gunakan dan bukan nilainya. Apabila kita eksekusi programnya maka tampilannya akan tampak seperti pada Gambar 9.4.

```

C:\TestCode\C++\test-02\Debug\test-02.exe
Nilai X : 5
Nilai *Alamat : 5
Nilai Alamat : 0x22ff88
Nilai &X : 0x22ff88
Nilai X : 20
Nilai *Alamat : 20
Nilai Alamat : 0x22ff88
Nilai &X : 0x22ff88

Terminated with return code 0
Press any key to continue ...

```

Gambar 9.4. Hasil eksekusi deklarasi pointer.

Perhatikan nilai-nilai output pada Gambar 9.4. Nilai seperti 0x22ff88 adalah angka hexadesimal dari alamat variabel. Kalau kalian amati, ketika kita

memasukkan nilai 5 pada variabel X maka variabel \*Alamat akan berisi juga nilai 5. Demikian juga ketika kita memasukkan nilai 20 pada variabel \*Alamat, nilai X juga berubah menjadi 20. Hal ini karena variabel \*Alamat dan X menempati alamat memori yang sama.

Setiap kali kita mendeklarasikan sebuah *pointer*, maka otomatis *pointer* akan menunjuk alamat acak pada memori. Oleh karena itu kita harus mengeset variabel *pointer* tersebut agar tidak menunjuk alamat tertentu dengan cara memberi nilai NULL. Perhatikan contoh kode berikut dan hasil eksekusinya (Gambar 9.5).

Contoh 9.21. Mendeklarasikan pointer dengan NULL.

```
#include <iostream>

using namespace std;

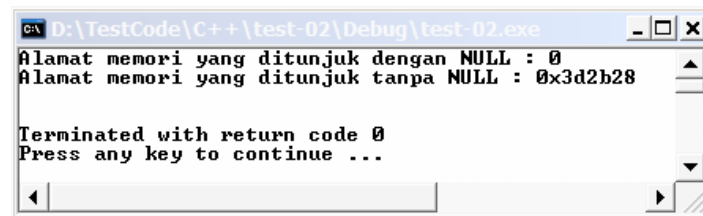
int main() {

    long *Alamat;
    long *Alamat1;

    Alamat = NULL;

    cout<<"Alamat memori yang ditunjuk dengan NULL :
"<<Alamat<<endl;
    cout<<"Alamat memori yang ditunjuk tanpa NULL :
"<<Alamat1<<endl;

    return 0;
}
```



Gambar 9.5. Hasil eksekusi pointer NULL.

### 9.3.2. Penggunaan *New* dan *Delete* pada Pointer

Kata kunci *New* dapat digunakan untuk mengalokasikan memori pada ruang yang masih kosong. Kata kunci ini diikuti oleh tipe data yang akan dialokasikan sehingga *compiler* akan mengetahui seberapa besar memori yang akan dialokasikan. Sedangkan *delete* merupakan kebalikan dari *new*. *Delete* akan membebaskan memori dari variabel yang kita gunakan. Perintah *delete* ini sangat penting, karena apabila kita tidak membebaskan memori dari

penggunaan maka akan sangat memboroskan pemakaian memori dan pada akhirnya akan membuat sistem berjalan tidak semestinya. Perhatikan contoh penggunaan *new* dan *delete* berikut ini.

Contoh 9.22. Penggunaan *new* dan *delete*.

```
#include <iostream>

using namespace std;

int main() {
    int *Alamat;

    // Melakukan alokasi memori
    Alamat = new int;

    // Menggunakan memori yang telah dialokasikan
    *Alamat = 100;

    cout<<"Nilai *Alamat : "<<*Alamat<<endl;

    // membebaskan memori
    delete Alamat;

    return 0;
}
```

### 9.3.3. Array

Seperti telah dijelaskan pada bab-bab sebelumnya, array dapat digunakan untuk menyimpan banyak data yang tipenya sama pada satu nama variabel. Pendefinisian array pada C++ hamper mirip dengan VB atau Java. Hanya sintaks penulisannya yang berbeda. Perhatikan contoh array sederhana berikut ini.

Contoh 9.23. Array sederhana.

```
#include <iostream>

using namespace std;

int main() {
    // deklarasi array A dengan 5 buah elemen bertipe int
    int A[5];

    cout<<"A[C]"<<"    "<<"B"<<endl;
    // Memasukkan nilai ke dalam elemen array
    for (int C=0; C<5; C++) {
        double B = 5;
        A[C] = C;
        B = A[C]/B;
        cout<<"A[C]"<<"    "<<"B"<<endl;
    }
```

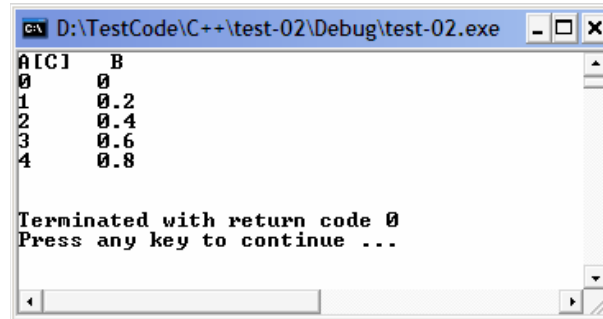


```

    }
    return 0;
}

```

Pada contoh di atas kita mendeklarasikan array dengan nama A dan tipe datanya adalah int. C pada kode di atas adalah variabel *counter*. Untuk memasukkan nilai pada array, perhatikan pada baris `A[C] = C`. C di sini adalah indeks dari variabel A. Sama seperti halnya pada VB ataupun Java, nilai default awal dari indeks array adalah 0. Apabila contoh di atas dijalankan, maka output akan tampak seperti pada Gambar 9.6.



Gambar 9.6. Output hasil eksekusi program array sederhana.

Pada contoh 9.23 di atas kita mendeklarasikan array tanpa inisialisasi nilai. Tapi sebenarnya kita dapat langsung member nilai bersamaan dengan pendeklarasian variabel array. Kemudian kita dapat merubah nilainya ketika program sedang berjalan atau dibagian lain dari program. Perhatikan contoh 9.24 berikut.

Contoh 9.24. Deklarasi dan inisialisasi array.

```

#include <iostream>

using namespace std;

int main() {
    // Deklarasikan dan inisialisasi array
    char X[3] = { 'A', 'B', 'C' };

    // Menampilkan nilai awal elemen array
    cout<<"Nilai array awal"<<endl;
    cout<<"X[0] = "<<X[0]<<endl;
    cout<<"X[1] = "<<X[1]<<endl;
    cout<<"X[2] = "<<X[2]<<endl;

    // Mengubah elemen ke-1 dan ke-2
    X[0] = 'G';
    X[1] = 'H';
}

```

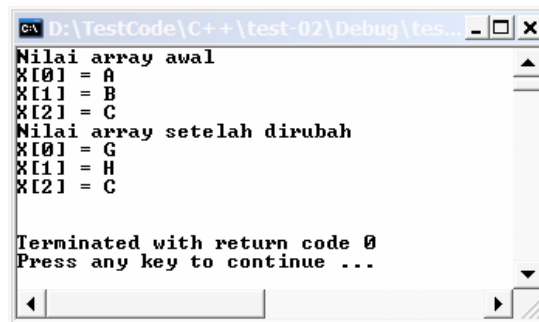
```

// Menampilkan nilai perubahan dari elemen array
cout<<"Nilai array setelah dirubah"<<endl;
cout<<"X[0] = "<<X[0]<<endl;
cout<<"X[1] = "<<X[1]<<endl;
cout<<"X[2] = "<<X[2]<<endl;

return 0;
}

```

Jika kita jalankan kode program di atas, maka hasilnya akan tampak seperti pada Gambar 9.7.



Gambar 9.7. Hasil eksekusi deklarasi dan inisialisasi array.

C++ juga menyediakan fitur untuk membuat array multidimensi. Deklarasi array multidimensi dapat dilakukan dengan cara seperti pada contoh 9.25.

Contoh 9.25. Deklarasi array multidimensi.

```

#include <iostream>

using namespace std;

int main() {

    // Deklarasi dan inisialisasi array multidimensi
    int Matrik_A[3][2] = { {1,2}, {3,4}, {5,6} };
    int Matrik_B[2][3] = { {1,2,3}, {4,5,6} };

    // Mendeklarasikan variabel untuk indeks pengulangan
    int j, k;

    // Menampilkan nilai yang tersimpan dalam elemen array A
    cout<<"ISI MATRIKS A"<<endl;
    for (j=0; j<3; j++) {
        for (k=0; k<2; k++) {

```

```

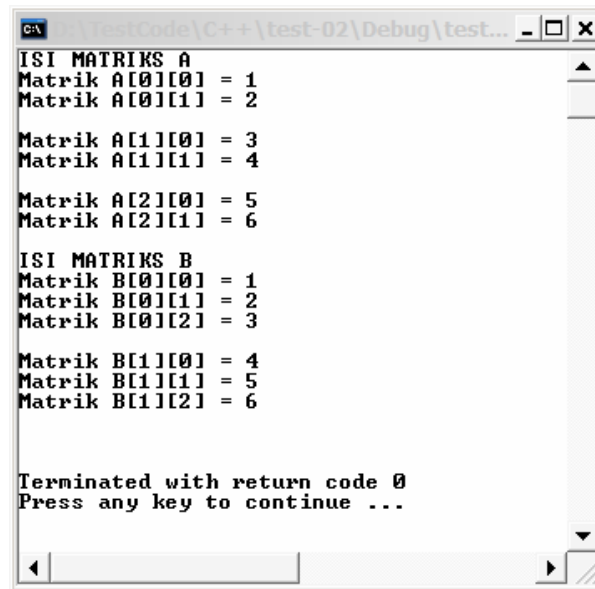
        cout<<"Matrik A["<<j<<"]["<<k<<"] =
"<<Matrik_A[j][k]<<endl;
    }
    cout<<endl;
}

// Menampilkan nilai yang tersimpan dalam elemen array B
cout<<"ISI MATRIKS B"<<endl;
for (j=0; j<2; j++) {
    for (k=0; k<3; k++) {
        cout<<"Matrik B["<<j<<"]["<<k<<"] =
"<<Matrik_B[j][k]<<endl;
    }
    cout<<endl;
}

return 0;
}

```

Pada contoh di atas, ada dua buah array multidimensi yaitu Matrik\_A dan Matrik\_B. Keduanya merupakan matriks dua dimensi. Matrik\_A memiliki 3 baris dan 2 kolom sedangkan Matrik\_B memiliki 2 baris dan 3 kolom. Pengisian nilai elemen array dilakukan pada saat deklarasi. Namun kalian dapat mengisi dengan menggunakan masukan dari keyboard melalui perintah cin. Hasil eksekusi dari program di atas akan tampak seperti pada Gambar 9.8.



```

C:\testCode\C++\test-02\Debug\test...
ISI MATRIKS A
Matrik A[0][0] = 1
Matrik A[0][1] = 2

Matrik A[1][0] = 3
Matrik A[1][1] = 4

Matrik A[2][0] = 5
Matrik A[2][1] = 6

ISI MATRIKS B
Matrik B[0][0] = 1
Matrik B[0][1] = 2
Matrik B[0][2] = 3

Matrik B[1][0] = 4
Matrik B[1][1] = 5
Matrik B[1][2] = 6

Terminated with return code 0
Press any key to continue ...

```

Gambar 9.8. Hasil eksekusi array multidimensi.

## 9.4. KELAS

Konsep kelas pada C++ sama persis dengan apa yang telah kalian pelajari pada bab 8. Konsep ini diturunkan dari paradigma pemrograman berorientasi obyek yang telah disampaikan pada bab tersebut. Apabila kalian telah mengerti dengan baik konsep pemrograman berorientasi obyek dan pemrograman kelas pada Java, maka konsep kelas pada C++ bukanlah perkara yang sulit. Perlu kalian ketahui, Java mengambil sebagian besar konsep pemrograman C++ (termasuk kelas) untuk diterapkan. Perbedaan utama mungkin hanya pada sintaks penulisannya. Pada bab ini konsep pemrograman berorientasi obyek tidak akan dibahas ulang, namun akan langsung pada penerapannya pada C++.

### 9.4.1. Deklarasi Kelas

Sama seperti pada Java, pembuatan kelas dalam C++ menggunakan kata kunci `class`. Di dalam kelas tersebut terdapat data dan method yang akan digunakan oleh obyek yang akan dibuat (instance) dari kelas tersebut. Data dan method ini biasan disebut sebagai anggota kelas (class member). Method dalam C++ sama bentuknya dengan fungsi yang telah kalian pelajari di sub bab sebelumnya. Perhatikan contoh berikut.

Contoh 9.26. Deklarasi dan penggunaan kelas.

```
#include <iostream>
using namespace std;

class PersegiPanjang {
    int x, y;
public:
    void set_nilai (int,int);
    int luas() {return (x*y);}
};

void PersegiPanjang::set_nilai (int a, int b) {
    x = a;
    y = b;
}

int main () {
    PersegiPanjang pp1, pp2;
    pp1.set_nilai (3,4);
    pp2.set_nilai (7,12);
    cout << "Luas pp1 : " << pp1.luas()<<endl;
    cout << "Luas pp2 : " << pp2.luas()<<endl;
    return 0;
}
```

Pada contoh di atas, kelas yang kita deklarasikan bernama `PersegiPanjang` dan mempunyai anggota kelas dua data yaitu `x` dan `y` dan dua *method* yaitu `set_nilai` dan `luas()`. Dua buah *method* tersebut ditetapkan mempunyai akses *public*. Seperti halnya java, ada 3 hak akses terhadap data atau method dalam

kelas, yaitu *public*, *private* dan *protected*. *Public* berarti anggota kelas tersebut dapat diakses dari luar kelas. *Private* berarti anggota kelas tersebut hanya dapat diakses di dalam kelas tersebut. Sedangkan *protected* berarti anggota kelas tersebut dapat diakses oleh turunan (*subclass*) dari kelas tersebut, tetapi tidak oleh bagian di luar kelas.

*Method* `set_nilai` mempunyai dua argument/parameter yang semuanya bertipe data `int` namun tidak memiliki pengembalian nilai, sehingga kita menggunakan kata kunci *void*. *Method* `luas()` tidak memiliki argumen namun memiliki pengembalian nilai. Untuk *method* yang memiliki pengembalian nilai maka kita menggunakan tipe data di depan nama *method*. Bukalah kembali bab 8 untuk mengingat kembali tentang argumen dalam *method*.

Pada C++ kita dapat mengimplementasikan *method* di dalam kelas atau di luar kelas, tetapi deklarasi *method* harus berada di dalam kelas. Umumnya kita meletakkan implementasi *method* di luar kelas (di luar tanda { }). Perhatikan pada contoh di atas. Deklarasi *method* `set_nilai` terletak di dalam kelas `PersegiPanjang` namun implementasinya berada di luar kelas. Sedangkan *method* `luas()`, baik deklarasi maupun implementasi berada di dalam kelas. Pada implementasi *method* di luar kelas kita menggunakan tanda `::` untuk mendefinisikan anggota kelas di luar kelasnya (perhatikan pada baris `void PersegiPanjang::set_nilai (int a, int b)`).

Setelah kelas terbentuk kita dapat menggunakan dengan membuat obyek yang merupakan *instance* dari kelas tersebut. Perhatikan pada bagian yang diawali dengan `int main()`. Pada bagian ini kita membentuk dua obyek dengan nama `pp1` dan `pp2` sebagai *instance* dari kelas `PersegiPanjang`. Kemudian kita menggunakan obyek-obyek ini untuk memanggil *method* `set_nilai` dan `luas()` dari kelas `PersegiPanjang`. Ketikkan program di atas kemudian jalankan. Bagaimanakah hasilnya?

Seperti pada Java, umumnya suatu kelas akan mempunyai *constructor* yang digunakan untuk menginisialisasi variabel atau mengalokasikan memori. *Constructor* ini mempunyai nama yang sama dengan kelasnya. Di dalam suatu kelas sebaiknya juga dibentuk *method destructor*. *Destructor* adalah kebalikan dari *constructor*. Tujuan utamanya adalah mengembalikan nilai variabel ke bentuk awal dan membebaskan memori dari penggunaan variabel. *Method destructor* mempunyai nama yang sama dengan nama kelasnya, tetapi dengan ditambah awalan tanda `~`. Perhatikan contoh penggunaan *constructor* dan *destructor* berikut.

Contoh 9.27. Constructor dan destructor.

```
#include <iostream>
using namespace std;

class PersegiPanjang {
    int *panjang, *lebar;
public:
    PersegiPanjang (int,int);
    ~PersegiPanjang ();
};
```

```

        int luas () {return (*panjang * *lebar);}
    };

PersegiPanjang::PersegiPanjang (int a, int b) {
    panjang = new int;
    lebar = new int;
    *panjang = a;
    *lebar = b;
}

PersegiPanjang::~~PersegiPanjang () {
    delete panjang;
    delete lebar;
}

int main () {
    PersegiPanjang pp1 (3,4), pp2 (5,6);
    cout << "Luas pp1: " << pp1.luas() << endl;
    cout << "Luas pp2: " << pp2.luas() << endl;
    return 0;
}

```

#### 9.4.2. Inheritance

C++ juga memberikan fasilitas *inheritance* pada kelas. Proses pewarisan pada C++ agak lebih rumit dibandingkan dengan Java. Hal ini karena C++ memberikan kemungkinan pewarisan dengan pertimbangan hak akses. Ada dua hak akses dalam pewarisan *superclass* ke *subclass*, yaitu *public* dan *private*.

Apabila suatu kelas diturunkan sebagai *public* dari *superclass*nya maka ketentuannya adalah sebagai berikut:

- Bagian *public* yang ada pada *superclass* akan tetap menjadi *public* pada *subclass*.
- Bagian *protected* yang ada pada *superclass* akan tetap menjadi *protected* pada *subclass*nya.
- Bagian *private* yang ada pada *superclass* tidak akan dapat diakses oleh *subclass*.

Apabila suatu kelas diturunkan sebagai *private* dari *superclass*nya maka ketentuannya adalah sebagai berikut:

- Bagian *public* yang ada pada *superclass* akan menjadi *private* pada *subclass*.
- Bagian *protected* yang ada pada *superclass* akan menjadi *private* pada *subclass*nya.
- Bagian *private* yang ada pada *superclass* tidak akan dapat diakses oleh *subclass*.

Perhatikan contoh berikut.

Contoh 9.28. Pewarisan.

```
#include <iostream>
using namespace std;

class CPolygon {
protected:
    int width, height;
public:
    void set_values (int a, int b)
        { width=a; height=b;}
};

class CRectangle: public CPolygon {
public:
    int area ()
        { return (width * height); }
};

class CTriangle: public CPolygon {
public:
    int area ()
        { return (width * height / 2); }
};

int main () {
    CRectangle rect;
    CTriangle trgl;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;
    return 0;
}
```

Pada kode program di atas, CPolygon adalah *superclass*, sedangkan CRectangle dan CTriangle adalah *subclass*. Pada kelas CPolygon, variabel *width* dan *height* dideklarasikan sebagai *protected*, karena ditujukan untuk bisa diakses oleh *subclass*nya saja. Selain itu kelas ini juga mempunyai *method* *set\_values*. Kedua variabel dan *method* ini akan diwariskan pada *subclass*nya yaitu CRectangle dan CTriangle. Perhatikan bagaimana CRectangle dan CTriangle dideklarasikan sebagai kelas turunan dari CPolygon dengan menggunakan kata kunci *public*. Sekarang coba ganti kata *public* pada deklarasi kelas CTriangle sehingga menjadi **class CTriangle: private CPolygon**. Apabila kalian kompilasi maka kalian akan menjumpai pesan kesalahan sebagai berikut:

Compiling source file(s)...

oo-test.cpp

oo-test.cpp: In function `int main()':

oo-test.cpp:9: error: `void CPolygon::set\_values(int, int)'  
is inaccessible

oo-test.cpp:28: error: within this context

```
oo-test.cpp:28: error: `CPolygon' is not an accessible base  
of `CTriangle'
```

Mengapa kesalahan kompilasi bisa terjadi? Hal ini karena berlakunya aturan di atas. *Method* `set_values` pada kelas `CPolygon` dideklarasikan dengan *public*, tetapi diturunkan ke kelas `CTriangle` dengan *private*. Hal ini akan merubah method yang semula *public* menjadi *private* ketika berada pada kelas `CTriangle`. Tentunya kalian ingat bila anggota kelas diberi hak akses *private* maka dia tidak dapat diakses dari luar.

### 9.4.3. Polimorfisme

Pada C++, untuk dapat menerapkan polimorfisme maka kita perlu menggunakan fungsi khusus yang dikenal sebagai fungsi *virtual*. Fungsi ini kita letakkan pada *superclass*, kemudian fungsi tersebut dapat kita definisikan ulang pada *subclass*. Perhatikan contoh berikut.

Contoh 9.29. Penggunaan fungsi virtual.

```
#include <iostream>

using namespace std;

class AnggotaSekolah {
    char* nama;
    char* alamat;
public:
    void SetNama(char* N) {
        nama = N;
    }
    void SetAlamat(char* A) {
        alamat = A;
    }
    char* GetNama() {
        return nama;
    }
    char* GetAlamat() {
        return alamat;
    }
    // Membuat fungsi virtual
    virtual void Bekerja() {
        cout<<"Bekerja"<<endl;
    }
    virtual void Berpakaian() {
        cout<<"Berpakaian"<<endl;
    }
};

class Siswa: public AnggotaSekolah {
    char* Jurusan;
    char* Program;
```



```

    int semester;
public:
    void SetJurusan(char* J) {
        Jurusan = J;
    }
    void SetProgram(char* P) {
        Program = P;
    }
    void SetSemester(int smt) {
        semester = smt;
    }
    char* GetJurusan() {
        return Jurusan;
    }
    char* GetProgram() {
        return Program;
    }
    int GetSemester() {
        return semester;
    }
    // override pada fungsi Bekerja
    void Bekerja() {
        cout<<"Bekerja menuntut ilmu"<<endl;
    }
    // override pada fungsi Berpakaian
    void Berpakaian() {
        cout<<"Berpakaian seragam putih abu-abu"<<endl;
    }
};

class Guru: public AnggotaSekolah {
    char* jabatan;
    char* keahlian;
public:
    void SetJabatan(char* jbt) {
        jabatan = jbt;
    }
    void SetKeahlian(char* khl) {
        keahlian = khl;
    }
    char* GetJabatan() {
        return jabatan;
    }
    char* GetKeahlian() {
        return keahlian;
    }
    // override pada fungsi Bekerja
    void Bekerja() {
        cout<<"Bekerja mengajarkan ilmu"<<endl;
    }
    // override pada fungsi Berpakaian

```

```

    void Berpakaian() {
        cout<<"Berpakaian baju seragam dinas resmi"<<endl;
    }
};

// Fungsi utama
int main() {

    // instansiasi pada kelas AnggotaSekolah, Siswa dan Guru
    AnggotaSekolah As;
    Siswa Sw;
    Guru Gr;

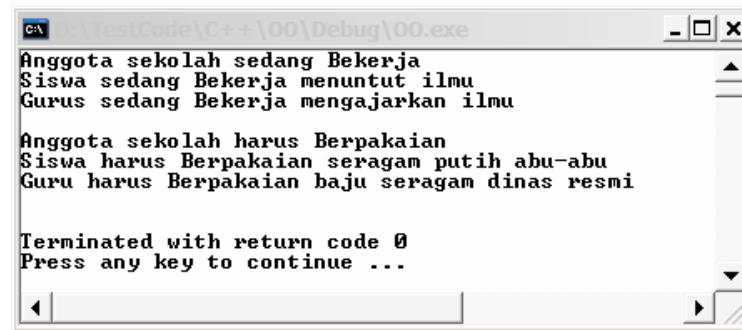
    // Memanggil fungsi Bekerja dari masing-masing kelas
    cout<<"Anggota sekolah sedang ";
    As.Bekerja();
    cout<<"Siswa sedang ";
    Sw.Bekerja();
    cout<<"Guru sedang ";
    Gr.Bekerja();
    cout<<'\n';

    // Memanggil fungsi Berpakaian dari masing-masing kelas
    cout<<"Anggota sekolah harus ";
    As.Berpakaian();
    cout<<"Siswa harus ";
    Sw.Berpakaian();
    cout<<"Guru harus ";
    Gr.Berpakaian();

    return 0;
}

```

Pada kode program di atas, ada dua fungsi/*method virtual* yaitu Bekerja dan Berpakaian. Method inilah yang akan kita gunakan pada *subclass* namun dengan penerapan yang lain. Perhatikan isi dari masing-masing method tersebut pada masing-masing *subclass*. Cara ini biasa disebut sebagai *overriding*. Coba kembali bab 8 untuk memperjelas pengertian *overriding*. Bandingkan juga bagaimana *overriding* dilakukan pada Java dan C++. Jika program di atas dijalankan, maka hasilnya akan tampak seperti pada Gambar 9.9. *Overloading* juga dapat dilakukan pada fungsi virtual. Kalian tentu masih ingat perbedaan *overriding* dan *overloading* yang sudah dijelaskan di bab 8.



Gambar 9.9. Hasil eksekusi fungsi virtual dan overriding.

Pada contoh 9.29 di atas, fungsi virtual dibuat lengkap dengan isi dari fungsi tersebut. Namun sebenarnya C++ juga menyediakan fungsi virtual murni (*pure virtual function*) yang hanya ada deklarasi fungsinya tapi tidak ada isinya. Konsep ini mirip dengan ketika kalian pelajari tentang *interface* pada Java. Fungsi virtual murni ini kemudian akan diterjemahkan isinya pada kelas-kelas yang merupakan turunan dari kelas tersebut. Keuntungan dari penggunaan fungsi virtual murni ini adalah keleluasaan kita untuk mendefinisikan fungsi-fungsi tersebut pada kelas turunannya. Fungsi virtual murni biasanya digunakan pada kelas abstrak. Kelas abstrak adalah kelas yang mempunyai paling tidak satu fungsi virtual murni. Karena masih abstrak kita tidak diperbolehkan untuk membuat obyek langsung dari kelas abstrak.

Konsep polimorfisme pada C++ disusun dengan berdasarkan pengertian pada fungsi virtual, fungsi virtual murni, *overriding*, *overloading*, dan kelas abstrak. Perhatikan contoh polimorfisme berikut ini.

Contoh 9.30. Penerapan polimorfisme.

```
#include <iostream>
using namespace std;

class CPolygon {
protected:
    int width, height;
public:
    void set_values (int a, int b)
        { width=a; height=b; }
    virtual int area (void) =0; //fungsi virtual murni
    void printarea (void)
        { cout << this->area() << endl; }
};

class CRectangle: public CPolygon {
public:
    // overriding fungsi area
    int area (void)
        { return (width * height); }
```

```

};

class CTriangle: public CPolygon {
public:
    // overriding fungsi area
    int area (void)
    { return (width * height / 2); }
};

int main () {
    CRectangle rect;
    CTriangle trgl;
    CPolygon *ppoly1 = &rect;    // mendefinisikan obyek
pointer
    CPolygon *ppoly2 = &trgl;    // mendefinisikan obyek
pointer
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    ppoly1->printarea();
    ppoly2->printarea();
    return 0;
}

```

Pada contoh di atas, kelas CPolygon adalah kelas abstrak yang memiliki fungsi virtual murni yaitu area. Perhatikan cara mendeklarasikan fungsi virtual murni pada baris yang diawali dengan pernyataan virtual. Fungsi ini tidak dibuat isinya tapi dibuat dengan tanda = 0. Kita tidak dapat membuat obyek langsung dari kelas CPolygon ini. Tetapi kita dapat membuat obyek *pointer* untuk mengalokasikan memori berdasarkan kelas ini. Pada kelas CPolygon juga digunakan kata kunci *this*. Kata kunci ini berfungsi untuk menunjuk pada kelas itu sendiri. Pernyataan **this->area()** pada kode di atas sama artinya dengan **CPolygon->area()**. Jadi pernyataan ini sama artinya dengan memanggil fungsi virtual *area* di dalam kelas itu.

Pada kode diatas dibuat dua variabel *pointer* *\*ppoly1* dan *\*ppoly2* yang nilainya sama dengan nilai dari alamat variabel *rect* dan *trgl*. Perhatikan dengan baik penggunaan tanda \* dan & untuk merujuk pada alamat memori. Jalankan program di atas dan perhatikan hasilnya.

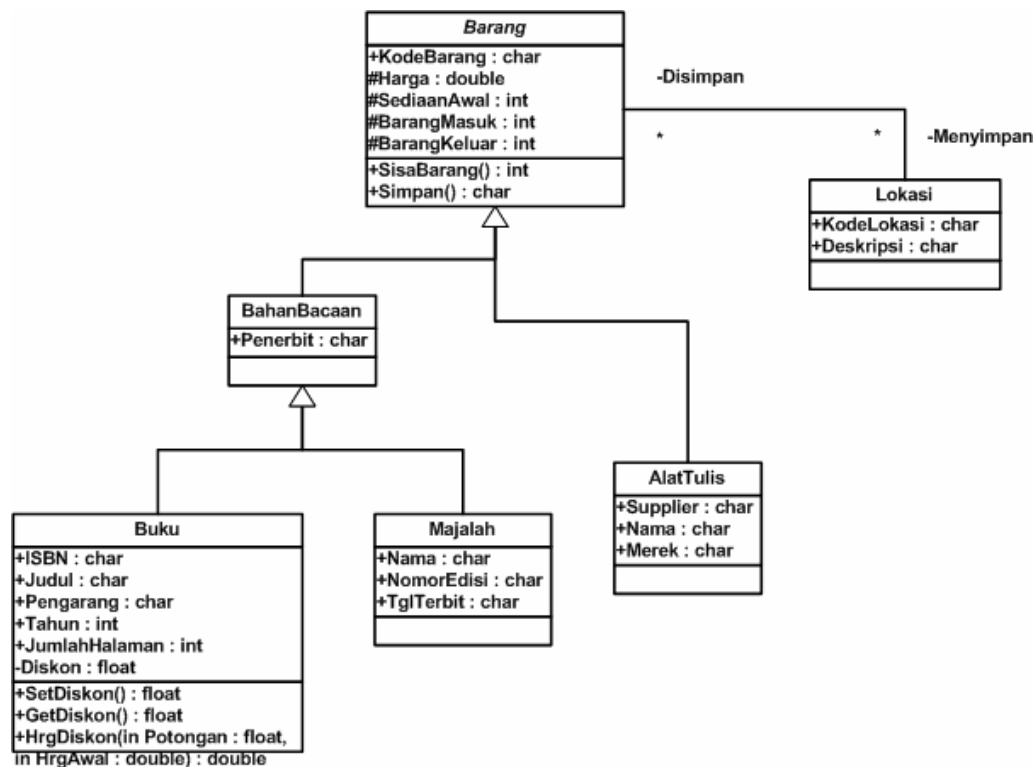
## 9.5. MERANCANG APLIKASI BERORIENTASI OBYEK

Konsep dasar tentang pemrograman berorientasi obyek telah kalian pelajari pada Bab 8. Sedangkan penerapannya dalam bahasa pemrograman juga telah kalian pelajari dengan menggunakan Java pada bab 8 dan C++ di bab ini. Pada bagian ini kita akan mencoba untuk membuat rancangan aplikasi berorientasi obyek.

Kasus yang kita angkat adalah salah satu bagian dari aplikasi penjualan pada toko buku, yaitu persediaan barang. Jika kalian cermati ada banyak jenis barang yang dijual pada sebuah toko buku, antara lain buku-buku bacaan,

majalah, alat tulis kantor, dan lain-lain. Masing-masing barang mempunyai nama merek, harga, dan karakteristik yang lain. Selain itu masing-masing barang juga mempunyai ketentuan-ketentuan lain yang berhubungan dengan penetapan harga, potongan, jumlah pembelian dan tempat penyimpanan. Persediaan barang pada toko buku ditentukan oleh barang yang masuk dari hasil pemesanan dan barang yang keluar karena terjual. Setiap barang keluar maka secara otomatis barang di dalam persediaan akan berkurang.

Langkah awal dalam aplikasi berorientasi obyek adalah dengan melakukan abstraksi pada permasalahan. Kalau kita perhatikan dengan seksama pada kasus di atas, kita bisa membuat kelas induk dari semua jenis barang yang ada. Hal ini karena selain karakteristik khusus dari masing-masing jenis barang, juga ada karakteristik umum yang dimiliki oleh semua barang. Seperti contoh harga adalah karakteristik umum yang dimiliki oleh semua barang. Tetapi merek adalah khusus pada jenis item barang tertentu, terutama pada alat tulis. Demikian juga dengan karakteristi judul hanya dimiliki oleh buku bacaan. Dengan melihat kondisi tersebut kita dapat membuat abstraksi permasalahan sebagai berikut.



Gambar 9.10. Abstraksi kasus persediaan barang di toko buku.

Pada Gambar 9.10, kita membentuk 6 kelas untuk mempermudah kasus di atas. Ada dua macam hubungan yang terbentuk pada gambar di atas, yaitu hubungan pewarisan (inheritance) yang ditandai dengan tanda panah dan asosiasi (hubungan antara dua atau lebih kelas) yang ditandai garis tanpa tanda panah. Barang adalah superclass dan bisa kita nyatakan sebagai kelas abstrak. BahanBacaan dan AlatTulis adalah subclass dari Barang, sedangkan Buku dan Majalah adalah subclass dari BahanBacaan. Sedangkan Lokasi merupakan kelas yang menunjukkan tempat menyimpan barang.

Pada kasus di atas kita dapat menerapkan konsep inheritance dan polimorfisme serta menggunakan fungsi virtual murni pada kelas Barang. Kemudian fungsi-fungsi ini dapat kita turunkan untuk diimplementasikan pada subclass-subclassnya. Rancangan kode program akan tampak seperti pada table 9.3.

Dengan melihat diagram dan tabel 9.3, kita dapat menerapkan dalam kode program. Akan ada 6 kelas yang kita buat kodenya. Masing-masing kelas bisa dalam satu file sendiri-sendiri atau semua kelas bisa kita letakkan dalam satu file. Kemudian implementasi dari class-class tersebut kita buat dalam satu file tersendiri. Untuk menghubungkan antar file dan memanggil class pada file lain kita menggunakan preprocessor `#include` diikuti dengan nama filenya.

Tabel 9.3. Kelas, fungsi, dan parameter pada aplikasi persediaan toko buku.

<i><b>Nama Kelas</b></i>	<i><b>Variabel</b></i>	<i><b>Fungsi</b></i>	<i><b>Parameter</b></i>	<i><b>Keterangan</b></i>
Barang	KodeBrg  Harga SediaanAwal BrgKeluar BrgMasuk	- SisaBarang  - Simpan	pAwal, bKeluar, bMasuk kodeLok	Kelas Abstrak
BahanBacaan	Penerbit			Subclass dan mewarisi anggota kelas Barang
Buku	ISBN Judul Pengarang Tahun JumlahHalaman Diskon	- setDiskon - getDiskon - HrgDiskon	Hrg, Potongan	Subclass dari BahanBacaan dan mewarisi anggota kelas tersebut
Majalah	Nama NoEdisi TglTerbit			Subclass dari BahanBacaan dan mewarisi anggota kelas tersebut
AlatTulis	Supplier			Subclass dari

	Nama Merek			BahanBacaan dan mewarisi anggota kelas tersebut
Lokasi	KodeLokasi Deskripsi	setKodeLok getKodeLok		Kelas yang berasosiasi dengan kelas Barang

## 9.6. RINGKASAN

- Struktur umum program dalam C++ meliputi bagian pendaftaran file, pendefinisian fungsi, bagian main(), dan blok kode.
- Tipe data primitive pada C++ terdiri dari int, long, float, double, char, bool, dan short. Tipe data composite yang disediakan adalah struct, enum dan array.
- Tipe-tipe operator yang dapat digunakan adalah operator assignment, operator unary, operator binary, operator relasional, operator bitwise dan operator ternary.
- Struktur control pemilihan dapat dilakukan dengan *if* (tanpa *then*) dan *switch ... case*. Sedangkan untuk pengulangan, C++ menyediakan perintah *for*, *while*, dan *do-while*.
- Pernyataan input dan output dapat dilakukan dengan perintah cout dan cin sebagai *stream* yang termasuk dalam kelas *iostream*.
- Fungsi dapat tidak mengembalikan nilai tetapi juga dapat mengembalikan nilai. Parameter pada fungsi dapat berupa parameter input, parameter output atau dua-duanya.
- *Pointer* adalah variabel. Namun berbeda dengan variabel normal, *pointer* menyimpan alamat pada memori, bukan nilai yang kita masukkan.
- C++ mendukung konsep pemrograman berorientasi obyek melalui pembuatan kelas, inheritance, fungsi virtual, overriding, overloading dan polimorfisme.

### 9.7. SOAL-SOAL LATIHAN

1. Buatlah program C++ untuk mencari rata-rata 5 buah bilangan 34, 56, 91, 11, 22.
2. Buatlah program menggunakan C++ untuk menentukan harga barang setelah di diskon dengan tampilan sbb:

```
Masukkan harga barang/unit      : Rp. 20000
Jumlah barang yang dibeli        : 5
-----
Total Harga sebelum diskon      : Rp. 100000
Diskon (10%)                    : Rp. 10000
-----
Harga bersih                     : Rp. 90000
```

3. Seseorang punya rekening tabungan di bank sebesar Rp. 10.000,- (saldo awal). Selanjutnya ia dapat menyetor atau mengambil tabungannya. Buatlah program dengan C++ untuk keperluan transaksi di bank tsb. Tampilan menu utamanya sbb:

```
-----
PT. BANK ABC
-----
Saldo : ....
Menu Transaksi
1. Setor Tabungan
2. Ambil Tabungan
3. Exit
Pilihan menu (1/2/3) ? ...
Ketentuan:
Bank membuat kebijakan bahwa saldo minimum yang
harus disisakan di rekening adalah Rp. 10.000,-
```

Jika nasabah memasukkan angka 1 maka dia akan diminta memasukkan jumlah rupiah yang akan disetor. Jika nasabah memilih angka 2 maka nasabah diminta memasukkan jumlah rupiah yang akan ditarik. Jika jumlah penarikan mengakibatkan saldo kurang dari Rp. 10000 maka program akan menolak. (Petunjuk: gunakan pernyataan cin untuk mendapatkan input dari keyboard)

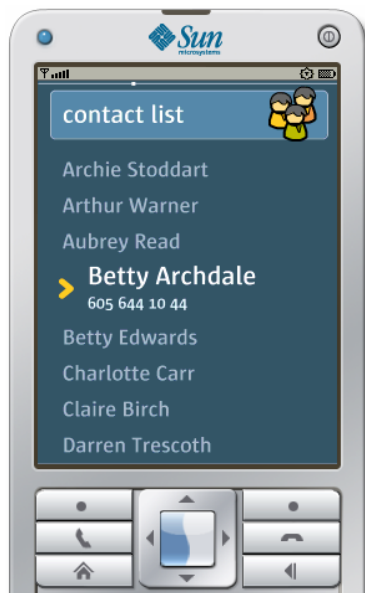
4. Buatlah program menggunakan function untuk menentukan nilai akhir suatu pelajaran. Terdapat 2 argumen function yaitu nilai ujian tengah semester dan nilai ujian akhir semester. Output yang diinginkan adalah jika nilai rata-rata lebih besar atau sama dengan 80 maka nilai akhirnya adalah A, jika nilai rata-rata kurang dari 80 dan lebih besar sama dengan 70 maka nilainya B, jika kurang dari 70 maka nilainya C.
5. Perhatikan contoh soal pada Bab 8 no 6. Buatlah diagram abstraksi sistemnya kemudian buatlah kode programnya dengan menggunakan bahasa C++.



---

## BAB 10 DASAR-DASAR SISTEM BASIS DATA

---



Gambar 10.1. Fasilitas contact list pada pesawat telepon seluler.

Gambar di samping ini adalah gambar sebuah pesawat telepon seluler atau lebih dikenal sebagai *handphone* yang sedang menjalankan salah satu aplikasi yaitu *Contact List*. Mungkin aplikasi seperti pada gambar ini tidak asing bagi kalian yang pernah menggunakan *handphone*. Aplikasi ini menyimpan nama teman, orang tua atau yang lainnya lengkap dengan nama dan nomor teleponnya. Bahkan kadang-kadang dilengkapi dengan alamat email, alamat kantor, nomor fax dan data-data lainnya.

Perhatikan aplikasi *Contact List* ini. Ketika kalian memasukkan nama dan nomor telepon seseorang maka nama dan nomor telepon tersebut tidak akan tertukar dengan nama atau nomor telepon orang lain. Semuanya teratur rapi. Kalian juga bisa mencari nama orang tertentu hanya dengan mengetikkan beberapa huruf yang ada hubungannya dengan nama. Perhatikan juga bahwa nama yang tersimpan selaluurut abjad, meskipun anda memasukkannya tidak berdasarkan urutan.

Data nama, nomor telepon dan data-data lainnya pada aplikasi *Contact List* disusun berdasarkan konsep pengaturan data yang lebih dikenal sebagai basis data. Pada bab ini kita akan mempelajari konsep-konsep dasar basis data. Kompetensi dasar sistem manajemen basis data (DBMS) merupakan bagian dari standar kompetensi membuat aplikasi berbasis Microsoft Access yang akan dibahas detil pada Bab 11. Bagian akhir dari bab ini akan ditutup dengan ringkasan dan latihan soal.

## TUJUAN

Setelah mempelajari bab ini diharapkan pembaca akan mampu :

- o Menjelaskan pengertian data, basis data dan sistem manajemen basis data (DBMS)
- o Menjelaskan *Entity-Relationship Diagram*
- o Menjelaskan basis data relasional

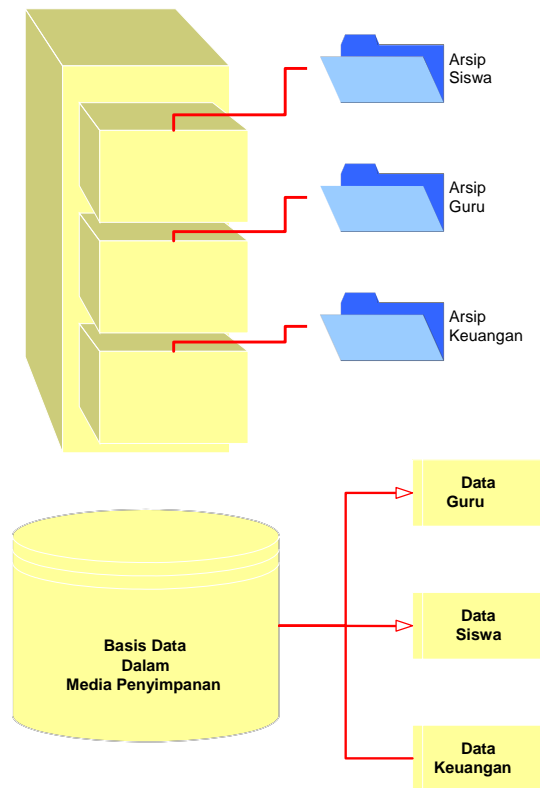
## 10.1. DATA, BASIS DATA DAN SISTEM MANAJEMEN BASIS DATA

### 10.1.1. Basis Data

Basis data (*database*) merupakan kumpulan dari data yang saling berhubungan satu dengan yang lainnya, tersimpan dalam perangkat keras komputer dan digunakan perangkat lunak untuk memanipulasinya. Basis data merupakan salah satu komponen utama dalam sistem informasi, karena merupakan basis dalam penyediaan informasi bagi para pemakai (Fathansyah, 1999; Post, 1999).

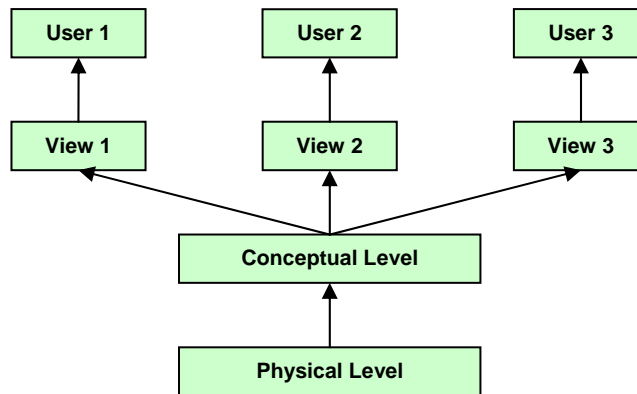
Jika dibayangkan, basis data mirip dengan lemari di ruang administrasi sekolah yang menyimpan berbagai arsip. Masing-masing jenis arsip dikelompokkan, diatur dan disimpan pada tempat yang telah ditentukan. Sehingga akan ada kelompok arsip siswa, arsip guru, arsip mata pelajaran, arsip keuangan, dan lain-lain. Perbedaannya hanya pada media penyimpanannya. Kalau lemari arsip menggunakan lemari dari kayu, besi atau plastik, sedangkan basis data menggunakan media penyimpan elektronis seperti disk (hard disc, CD, atau tape). Gambar 10.2 memberikan ilustrasi tentang kesamaan lemari arsip dan basis data.

Satu hal penting yang harus diperhatikan, basis data bukan hanya sekedar penyimpanan data secara elektronis. Tidak semua penyimpanan data elektronis bisa disebut basis data. Apabila penyimpanan itu tidak menggunakan prinsip pengaturan, pemisahan atau pengorganisasian maka kita tidak dapat menyebut penyimpanan data tersebut sebagai basis data. Pada Gambar 10.2 terlihat penerapan prinsip pengaturan, pengorganisasian atau pemisahan, baik pada lemari arsip atau pada basis data.



Gambar 10.2. Lemari arsip dan basis data.

Prinsip utama dalam basis data adalah konsep independensi data yaitu pemisahan data dari program aplikasinya (Lewis et al., 2002; Post, 1999). Sedangkan tujuan utama dalam basis data adalah membantu pengguna dalam abstraksi suatu sistem. Ada tiga level abstraksi yang biasanya digunakan yaitu *physical level*, *conceptual level* dan *view level* (Gambar 10.3). *Physical level* menunjukkan bagaimana data akan disimpan. *Conceptual level* berkaitan dengan data apa yang akan disimpan dan bagaimana hubungan antar data tersebut. *View level* merupakan level tertinggi yang menjelaskan bagian-bagian basis data pada pengguna tertentu (Ramakrishnan and Gehrke, 2000).



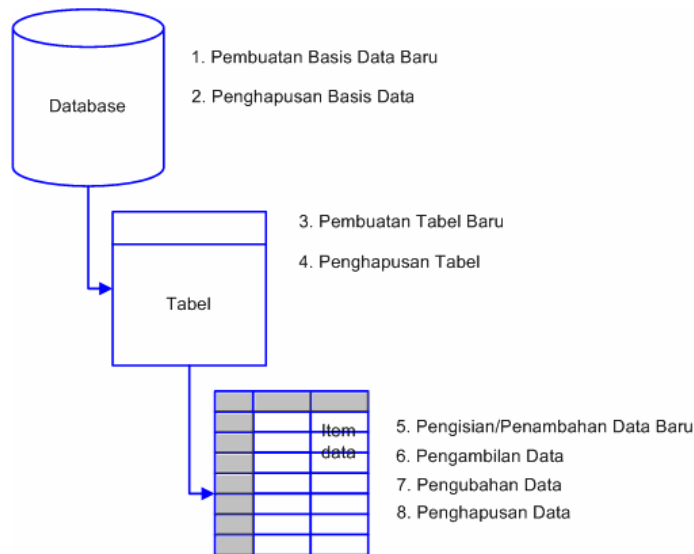
Gambar 10.3. Tingkatan dalam abstaksi data (Lewis et al., 2002).

Basis data mempunyai beberapa kriteria penting, yaitu :

1. Bersifat *data oriented* dan bukan *program oriented*.
2. Dapat digunakan oleh beberapa program aplikasi tanpa perlu mengubah basis datanya.
3. Dapat dikembangkan dengan mudah, baik volume maupun strukturnya.
4. Dapat memenuhi kebutuhan sistem-sistem baru secara mudah
5. Dapat digunakan dengan cara-cara yang berbeda.

Secara bertingkat, operasi dasar basis data dapat digambarkan dalam skema pada Gambar 10.4. Operasi-operasi tersebut meliputi:

- Pembuatan basis data baru (*create database*). Operasi ini sama dengan pembuatan atau pembelian lemari arsip yang baru.
- Penghapusan basis data (*drop database*). Operasi ini sama dengan pengrusakan atau penghancuran lemari arsip.
- Pembuatan tabel baru (*create table*). Operasi ini sama dengan penambahan kelompok arsip baru. Operasi ini baru bisa dijalankan jika basis data telah dibuat.
- Penghapusan tabel (*drop table*). Operasi ini sama dengan pengrusakan kelompok arsip lama. Operasi ini baru bisa dijalankan jika tabel telah ada pada suatu basis data.



Gambar 10. 4. Operasi-operasi dasar pada basis data.

- Pengisian atau penambahan data baru (*insert data*) pada suatu tabel. Operasi ini mirip dengan penambahan lembaran arsip baru pada kelompok arsip. Operasi ini baru bias dijalankan jika tabel telah dibuat.
- Pengambilan data dari suatu tabel (*retrieve data*). Operasi ini mirip dengan pencarian lembaran arsip yang tersimpan dalam kelompok arsip.
- Pengubahan data dari suatu tabel (*update data*). Operasi ini mirip dengan perbaikan isi lembaran arsip dari suatu kelompok arsip
- Penghapusan data dari suatu tabel (*delete*). Operasi ini mirip dengan penghapusan sebuah lembaran arsip dari suatu kelompok arsip.

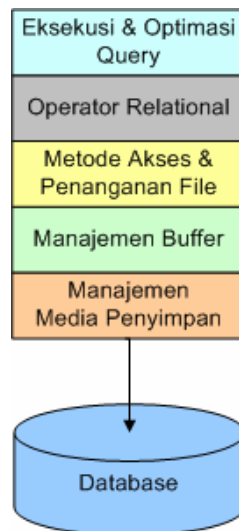
Basis data dibangun untuk memenuhi tujuan dalam pengorganisasian data, yang antara lain sebagai berikut :

1. Efisiensi meliputi kecepatan (*speed*), ruang simpan (*space*) dan keakuratan (*accuracy*).
2. Menangani data dalam jumlah besar.
3. Kebersamaan pemakaian (*Shareability*).
4. Meniadakan duplikasi dan inkonsistensi data.

### 10.1.2. Sistem Manajemen Basis Data

Pengelolaan basis data secara fisik tidak dilakukan oleh pemakai secara langsung, tetapi ditangani oleh sebuah Perangkat Lunak yang khusus/spesifik. Perangkat lunak inilah disebut DBMS yang akan menentukan bagaimana data diorganisasi, disimpan, diubah dan diambil kembali. Ia juga menerapkan

mekanisme pengaman data, pemakaian data secara bersama, pemaksaan keakuratan/konsistensi data, sebagainya. Secara ringkas struktur suatu DBMS dapat dilihat pada Gambar 10.5.



Gambar 10.5. Struktur umum DBMS.

Aplikasi-aplikasi tambahan bersifat opsional (bisa ada dan bisa tidak) dan biasanya terdapat pada DBMS sebagai fungsi tambahan. Sebagai contoh, aplikasi pembuat *report* (laporan), aplikasi untuk mendisain *form*, aplikasi untuk membuat diagram atau *chart*, aplikasi untuk monitoring sistem, dan aplikasi-aplikasi lainnya.

Ada puluhan bahkan mungkin ratusan perangkat lunak DBMS yang tersedia. Masing-masing dengan spesifikasinya sendiri-sendiri. Mulai dari yang sangat sederhana sampai yang paling kompleks. Pada bagian ini kita akan membahas 5 buah DBMS yang cukup familiar dikalangan pengguna DBMS, yaitu Microsoft Access, MySQL, Microsoft SQL Server, PostgreSQL, dan Oracle.

- **Microsoft Access**

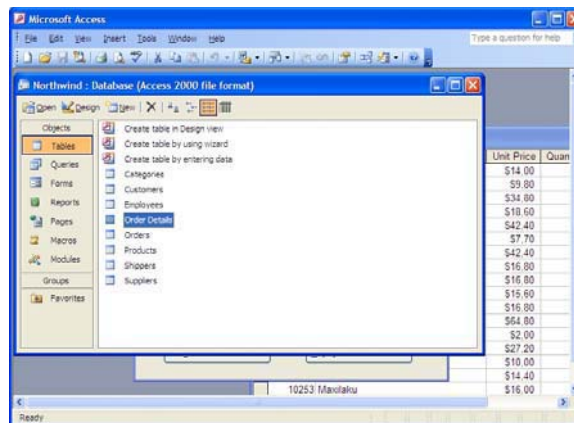


Gambar 10.6.  
Logo MS Access

Microsoft Access atau kadang disebut juga Microsoft Office Access adalah DBMS relational keluaran dari Microsoft yang termasuk dalam paket Microsoft Office. Microsoft Access mengkombinasikan engine relational Microsoft Jet Database, **Graphical User Interface** (GUI) dan perangkat pengembang perangkat lunak. Microsoft Access dapat menggunakan data yang disimpan dalam Microsoft Jet Database, Microsoft SQL Server, Oracle atau tipe lain asal kompatibel dengan ODBC (*Open Database Connectivity*).

Microsoft Access seringkali digunakan dalam

pengembangan aplikasi secara cepat (*Rapid Application Development*), terutama untuk membangun prototipe dan aplikasi stand-alone. Microsoft Access juga dapat digunakan sebagai basis data untuk aplikasi berbasis web sederhana. Namun pada aplikasi yang lebih kompleks, baik berdasarkan web atau tidak, Microsoft Access bukanlah pilihan yang baik. Terutama karena kekurangannya dalam menangani penggunaan oleh banyak pengguna (*multi-user*). Hal ini karena sebenarnya Microsoft Access adalah sebuah basis data personal yang lebih ditujukan untuk *single-user*. Microsoft Access juga tidak dilengkapi dengan *database triggers* dan *stored procedurs*.



Gambar 10.7. Tampilan Microsoft Access.

Salah satu keuntungan dari Microsoft Access bagi *programmer* adalah kompatibilitasnya terhadap SQL (*structured query language*) relatif tinggi. Pada Microsoft Access kita dapat membuat *query* berbasis teks atau berbasis GUI kemudian dapat langsung dieksekusi dengan mudah untuk mendapatkan hasil.

- MySQL

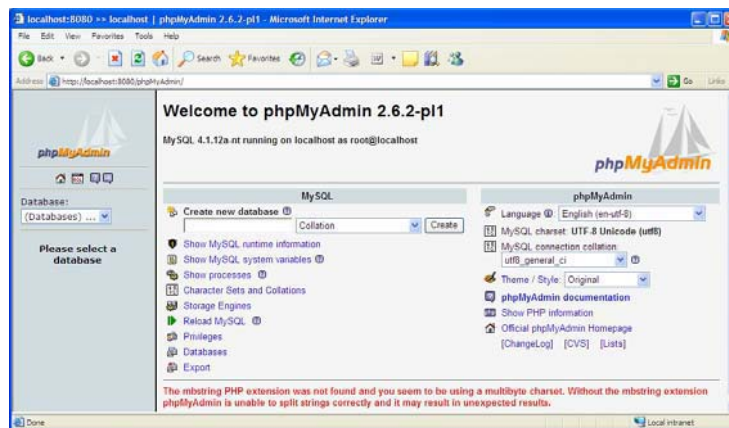


Gambar 10.8.  
Logo MySQL

MySQL adalah SQL-DBMS yang bersifat *multi-user* dan *multi-threaded*. MySQL berjalan sebagai *server* yang melayani banyak pengguna untuk mengakses sejumlah basis data. DBMS ini sangat populer di dunia aplikasi berbasis web sebagai komponen basis data. Selain karena tersedia dalam versi gratis, popularitas MySQL juga sangat dipengaruhi oleh populernya web server Apache dan bahasa pemrograman PHP. Istilah-istilah seperti LAMP (Linux-Apache-MySQL-PHP/Perl/Python), MAMP (Mac-Apache-MySQL-PHP/Perl/Python) dan WAMP (Windows-Apache-MySQL-PHP/Perl/Python) menjadi sangat terkenal. Banyak

sekali aplikasi berbasis web yang dibangun dengan menggunakan kombinasi perangkat lunak tersebut. WordPress, Drupal, Mambo, Wikipedia, PHP-Nuke, merupakan beberapa contoh aplikasi berbasis web yang menggunakan kombinasi ini.

Tidak seperti Microsoft Access, *default* instalasi MySQL tidak menyediakan GUI bagi pengguna untuk berinteraksi dengan basis data. Pengguna dapat berinteraksi dengan client yang menggunakan perintah-perintah berbasis teks. Namun saat ini telah banyak GUI yang dikembangkan untuk mempermudah interaksi dengan basis data, baik itu berupa aplikasi stand-alone (misalnya MySQL-Front, MySQL-GUI, dan lain-lain) atau yang berbasis web (misalnya, phpMyAdmin). Bahkan dengan menggunakan komponen MyODBC, MySQL dapat diakses dengan GUI dari Microsoft Access seperti halnya basis data yang kompatibel dengan ODBC lainnya.



Gambar 10.9. Tampilan awal phpMyAdmin.

- **Microsoft SQL Server**



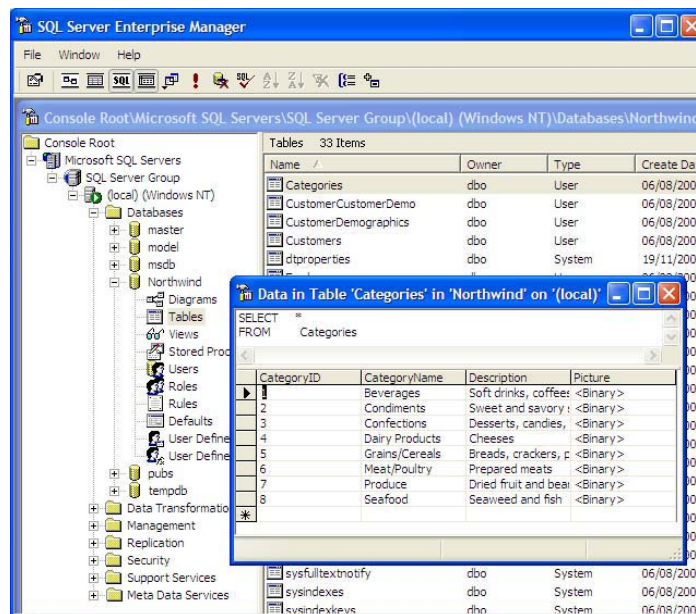
Gambar 10.10.  
Logo Microsoft  
SQL Server

Microsoft SQL Server adalah DBMS relational keluaran dari Microsoft seperti halnya Microsoft Access. Bahasa query utama yang digunakan adalah varian dari ANSI SQL yang disebut sebagai T-SQL (Transact-SQL). Bahasa ini membolehkan pengguna untuk membuat *stored procedure* sehingga meningkatkan efisiensi akses ke basis data. DBMS ini juga dilengkapi dengan fasilitas *clustering* dan *mirroring*. *Cluster* adalah kumpulan dari *server-server* yang konfigurasinya identik sehingga memungkinkan pembagian kerja antar *server*. Sedangkan fasilitas



*mirroring* membolehkan suatu DBMS untuk membuat tiruan (replika) dari isi basis data secara lengkap untuk digunakan pada server yang lain.

Microsoft SQL Server tersedia dalam beberapa versi distribusi. Pada Microsoft SQL Server yang didistribusikan bersama Microsoft Office atau Microsoft Visual Studio, biasa disebut MSDE (Microsoft SQL Server Database Engine), tidak dilengkapi dengan perangkat GUI. Pengguna dapat melakukan interaksi dengan menggunakan client berbasis perintah teks seperti halnya MySQL. Sedangkan pada versi yang lebih tinggi seperti versi personal atau profesional telah tersedia fasilitas GUI (Gambar 10.11).



Gambar 10.11. GUI pada Microsoft SQL Server

- PostgreSQL



Gambar 10.12.  
Logo PostgreSQL

PostgreSQL atau sering disebut Postgres termasuk dalam kategori *Object-Relational Database Management System* (ORDBMS). ORDBMS adalah DBMS yang selain menggunakan prinsip-prinsip basis data relational juga menggunakan pendekatan berorientasi obyek dalam model basis datanya. Postgres dikembangkan sebagai *free-software* dan bersifat terbuka (*open-source*) sehingga tidak dikendalikan oleh satu atau dua perusahaan.

Kelebihan Postgres dibandingkan DBMS lainnya

adalah, sifatnya yang *free* dan *open-source*, dukungan dokumentasinya yang luar biasa, fleksibilitasnya dan fitur-fiturnya yang tidak kalah dengan DBMS komersial. Selain mendukung model data *object-relational*, Postgres juga mendukung penggunaan basis data spasial (biasanya untuk penggunaan Sistem Informasi Geografis). Postgres juga mendukung operasi *multi-user* dan *multi-threaded*, bahkan mungkin lebih bagus dari MySQL dari sisi keamanan.

Seperti halnya MySQL dan Microsoft SQL Server, kita dapat berinteraksi dengan basis data pada Postgres menggunakan perintah-perintah disisi klien dengan tool yang disebut psql. Antar muka yang bersifat GUI juga telah banyak dikembangkan, diantaranya phpPgAdmin, PgAdmin, dan lain-lain.

- **Oracle Database**

**ORACLE**

Gambar 10.13.  
Logo Oracle

Nama Oracle Database atau Oracle RDBMS adalah nama yang sangat diperhitungkan dalam dunia DBMS. Oracle dikembangkan oleh Oracle Corporation.

Oracle menyimpan data secara logika dalam bentuk *tablespaces* dan secara fisik dalam bentuk file-file data. *Tablespaces* dapat berisi berbagai macam bagian memori, misalnya bagian data, bagian index dan lain sebagainya. Bagian-bagian ini berisi satu atau lebih area. Area-area ini berisi kumpulan blok data yang berdekatan. Oracle dapat menyimpan dan *store procedure* dan fungsi secara mandiri.

## 10.2. ENTITY-RELATIONSHIP DIAGRAM

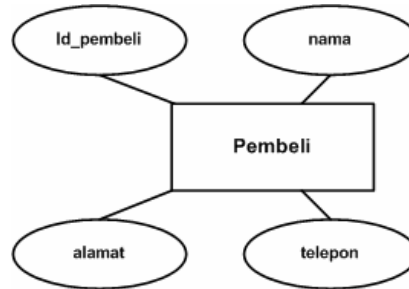
Model E-R biasa digambarkan dalam bentuk diagram yang disebut *Entity-Relationship Diagram* (ERD). ERD merupakan notasi grafis dalam pemodelan data konseptual yang digunakan untuk memodelkan struktur data dan hubungan antar data. Dengan ERD kita dapat menguji model dengan mengabaikan proses yang harus dilakukan. Dan dengan ERD kita mencoba menjawab pertanyaan seperti; data apa yang kita perlukan? bagaimana data yang satu berhubungan dengan yang lain? ERD menggunakan sejumlah notasi dan simbol untuk menggambarkan struktur dan hubungan antar data, pada dasarnya ada 3 macam simbol yang digunakan yaitu :

1. **Entitas** : Entitas digambarkan dalam bentuk persegi empat.



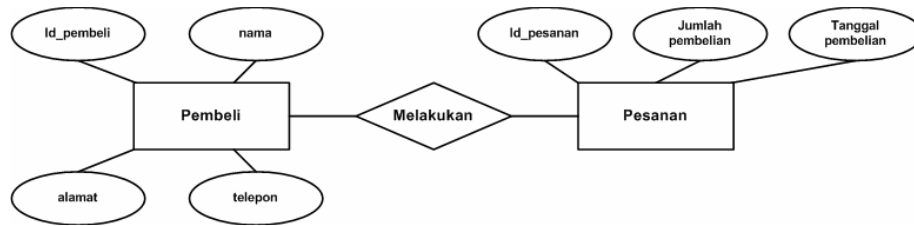
Gambar 10.14. Notasi entitas pada ER-Diagram

2. **Atribut** : Atribut digambarkan dalam bentuk *ellips* dan dihubungkan dengan entitas dimana atribut tersebut berada.



Gambar 10.15. Penggunaan notasi atribut pada ER-Diagram.

3. **Relationship**: Relationship digambarkan dalam bentuk intan/*diamonds*.



Gambar 10.16. Penggunaan notasi *relationship* pada ER-Diagram.

### 10.2.1. Entitas

Entitas adalah individu yang mewakili sesuatu yang nyata (eksistensinya) dan dapat dibedakan dari sesuatu yang lain. Dapat berupa suatu elemen dari suatu lingkungan, suatu sumber daya atau sebuah transaksi yang memiliki arti penting bagi suatu model yang akan dibangun.

Contoh Entitas set :

- o Semua Guru atau Guru saja.  
Himpunan ini memiliki anggota : Bapak Fahri, Ibu Fitri, Bapak Joko dan guru-guru yang lain.
- o Semua Siswa atau Siswa saja.  
Himpunan ini memiliki anggota : Joni, Ridho, Fanny, Donny dan siswa-siswa yang lain.
- o Semua Mobil atau Mobil saja.

Himpunan ini memiliki anggota : mobil Daihatsu, mobil Toyota, mobil Suzuki, dan mobil-mobil yang lain.

Mengidentifikasi ada atau tidaknya entitas dalam suatu masalah memang tidak mudah. Tapi biasanya apabila kita menjumpai kata benda dalam permasalahan tersebut maka kata tersebut biasanya merupakan kandidat entitas. Sebagai contoh bila kita akan membangun basis data perpustakaan sekolah, maka kita akan menjumpai buku, siswa, guru, petugas perpustakaan sebagai calon kuat entitas.

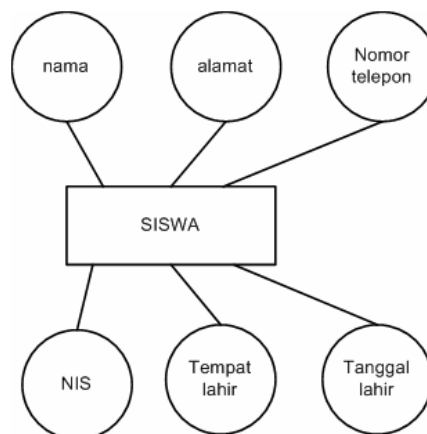
### 10.2.2. Atribut

Setiap entitas mempunyai atribut. Atribut adalah karakteristik atau ciri yang membedakan antara entitas satu dengan entitas yang lainnya.

Contoh Atribut :

- o Entitas Siswa.

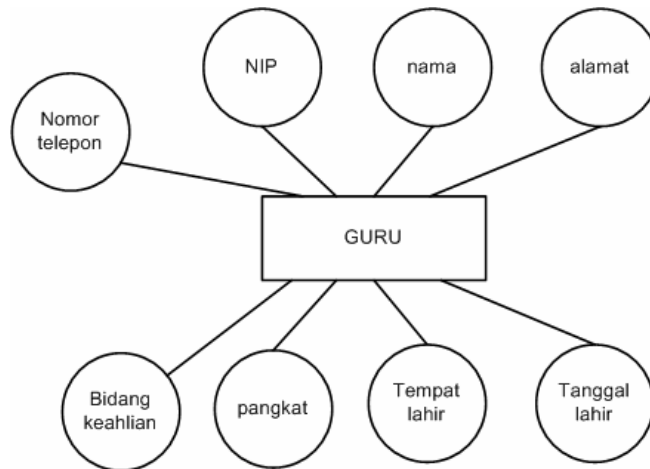
Memiliki atribut antara lain: nis (nomor induk siswa), nama, alamat, nomor telepon, tempat lahir, tanggal lahir dan lain-lain.



Gambar 10.17. Entitas siswa dan atributnya.

- o Entitas Guru.

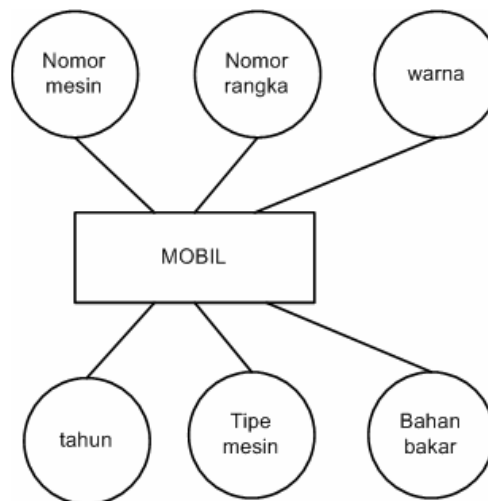
Memiliki atribut antara lain: NIP, nama, alamat, pangkat, nomor telepon, tempat lahir, tanggal lahir, bidang keahlian, dan lain-lain.



Gambar 10.18. Entitas guru dan atributnya.

- o Entitas Mobil.

Memiliki atribut antara lain: Nomor mesin, nomor rangka, warna, tahun keluar, tipe mesin, bahan bakar, dan lain-lain.



Gambar 10.19. Entitas mobil dan atributnya.

Tidak semua karakteristik dari entitas penting bagi suatu ruang lingkup masalah. Sebagai contoh pada masalah basis data perpustakaan, karakteristik nomor sepatu siswa bukanlah karakteristik yang penting yang dapat dijadikan sebagai atribut. Tetapi apabila ruang lingkup masalahnya adalah basis data pemesanan seragam sepatu siswa, maka nomor sepatu adalah atribut yang penting.

Selain memilih mana yang benar-benar penting bagi sebuah entitas, kita juga harus menentukan mana yang akan menjadi atribut kunci (*Primary Key*).

Pada contoh di atas (contoh atribut) kita dapat dengan mudah menentukan *primary key* dari entitas yang ada.

Contoh Atribut kunci (*Primary key*). :

- o Entitas Siswa.

Misalkan atribut yang dimiliki adalah : NIS (nomor induk siswa), nama, alamat, nomor telepon, tempat lahir, dan tanggal lahir. Dari keenam atribut ini, maka yang paling cocok menjadi *primary key* adalah NIS karena atribut ini yang paling unik. Tidak ada siswa yang memiliki NIS yang sama. Nama masih mungkin sama, tapi NIS tidak.

- o Entitas Guru.

Misalkan atribut yang dimiliki adalah: NIP, nama, alamat, pangkat, nomor telepon, tempat lahir, tanggal lahir, dan bidang keahlian. Atribut yang paling cocok menjadi *primary key* adalah NIP karena atribut ini yang paling unik. Tidak ada guru yang memiliki NIP yang sama. Sehingga NIP dapat dijadikan pengidentifikasi entitas guru.

- o Entitas Mobil.

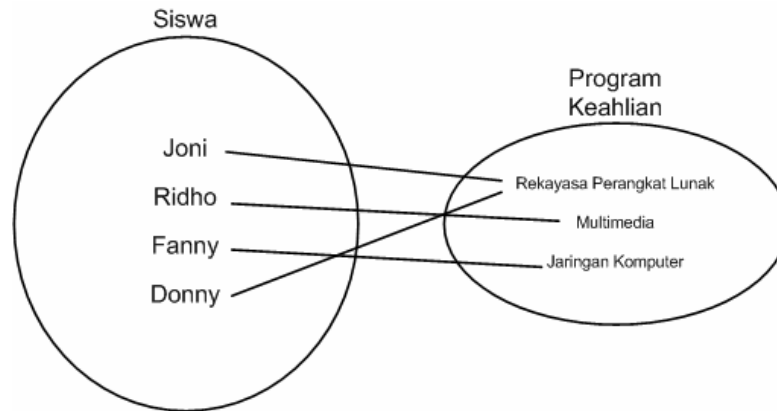
Misalkan atribut yang dimiliki adalah: Nomor mesin, nomor rangka, warna, tahun keluar, tipe mesin, dan bahan bakar. Di sini ada dua atribut yang unik yaitu *nomor mesin* dan *nomor rangka*. Pada kasus ini kita dapat memilih salah satu dari kedua atribut tersebut sebagai *primary key*.

Sebagai panduan, berikut ini merupakan ciri-ciri dari atribut yang dapat dipertimbangkan sebagai identifier (*candidate key*) :

- o Nilainya tidak berubah-ubah
- o Tidak mungkin berisi nilai null (kosong bukan nol)
- o Tidak berisi data nama atau lokasi yang mungkin berubah-ubah.

### 10.2.3. Relationship

*Relationship* atau relasi adalah hubungan yang terjadi antara sejumlah entitas. Misalkan dari entitas siswa ada seorang siswa yang memiliki *NIS* = "GHI007" dan *nama\_siswa* = "Donny" mempunyai relasi dengan entitas program keahlian dengan *kode\_program* = "RPL" dan *nama\_program* = "Rekayasa Perangkat Lunak". Relasi di antara kedua entitas mengandung arti siswa tersebut sedang mengambil program keahlian tersebut pada sekolah tertentu.



Gambar 10.20. *Relationship*.

Ramakrishnan and Gehrke (2000) menyebutkan bahwa konsep *relationship* pada model E-R berbeda dengan konsep *relation* di dalam model data relasional. *Relationship* adalah mekanisme yang menghubungkan antara entitas. Dalam implementasi ke dalam DBMS baik entitas maupun *relationship* akan direpresentasikan dalam bentuk tabel (*relation*).

Setiap *relationship* selalu mempunyai kardinalitas. Kardinalitas atau Derajat Relasi menunjukkan jumlah maksimum entitas yang dapat berelasi dengan entitas lain pada himpunan entitas yang lain.

Pada gambar 10.20 kita sebenarnya dapat melihat sebuah kardinalitas antara himpunan entitas siswa dengan himpunan entitas program keahlian. Siswa dapat berelasi hanya dengan satu entitas pada himpunan entitas program keahlian. Sebaliknya satu entitas pada program keahlian dapat berelasi dengan banyak siswa. Pada gambar tersebut terlihat Donny hanya dapat berhubungan dengan Rekayasa Perangkat Lunak, sedangkan Rekayasa Perangkat Lunak dapat berhubungan dengan Donny dan Joni.

Ada beberapa jenis tingkat hubungan (kardinalitas) antara entitas satu dengan entitas lainnya. Kardinalitas menunjukkan jumlah maksimum entitas pada suatu himpunan entitas yang dapat berelasi dengan entitas pada himpunan entitas yang lain. Secara umum ada tiga bentuk kardinalitas antar himpunan entitas, yaitu :

- o **satu ke satu (one to one)**. Hubungan dengan kardinalitas *one-to-one* artinya satu anggota suatu entitas set hanya boleh berhubungan dengan satu anggota entitas set yang lain. Hubungan antara entitas set suami dengan istri dapat dikelompokkan dalam hubungan *one-to-one*.



Gambar 10.21. Hubungan *one-to-one* suami dan istri.

- o **satu ke banyak / banyak ke satu (one to many / many to one).** Kardinalitas satu ke banyak dan banyak ke satu dapat dianggap sama karena tinjauan kardinalitas selalu dilihat dari dua sisi. Contohnya adalah pada suatu sekolah mempunyai aturan satu kelas terdiri dari banyak siswa tetapi tidak sebaliknya, yaitu satu siswa tidak dapat belajar pada kelas yang berbeda.



Gambar 10.22. Hubungan *one-to-many* kelas dengan siswa.

- o **banyak ke banyak (many to many).** Kardinalitas ini cukup rumit untuk dijelaskan namun seringkali kita jumpai. Misalnya hubungan siswa dengan mata pelajaran memiliki kardinalitas many-to-many. Siswa berhak mengambil (mempelajari) lebih dari satu matapelajaran dan setiap mata pelajaran boleh diambil (dipelajari) lebih dari satu siswa.

### 10.3. BASIS DATA RELASIONAL

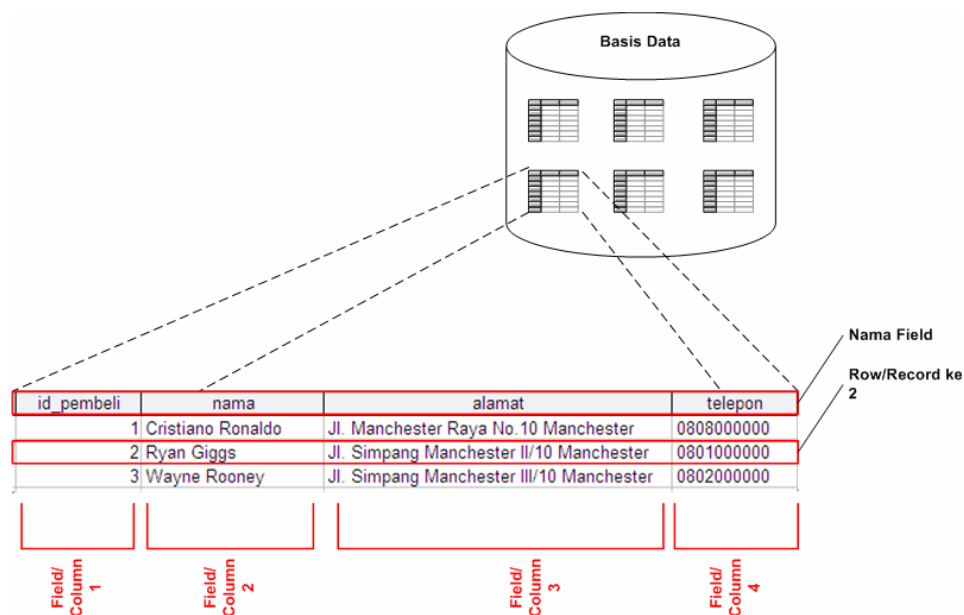
#### 10.3.1. Model Basis Data Relasional

Model basis data relasional diperkenalkan pertama kali oleh E.F. Codd pada tahun 1970. Model data ini didasarkan pada struktur matematis yang mudah dan alami, yaitu *relation* (tabel). Operasi-operasi manipulasi data semuanya berakar pada logika matematika. Hal ini menjadikan ekspresi-ekspresi pada tabel dapat dianalisis dan dioptimasi (Lewis et al., 2002).

Pembentuk utama dalam model data relasional adalah *relation* (tabel). *Relation* terdiri dari dua hal penting yaitu *schema* dan *instance*. *Relation instance* tidak lebih dari sebuah tabel dua dimensi dengan baris dan kolom. Baris (*row*) biasa disebut sebagai *tuple*, yang memiliki arti sama dengan *record* dalam suatu file. Tetapi berbeda dengan file record, semua *tuple* memiliki jumlah kolom yang sama dan tidak ada *tuple* dalam *relation instance* yang sama. Kolom dalam *relation instance* juga dikenal sebagai *attribute* atau *column* (Ramakrishnan and Gehrke, 2000; Lewis et al., 2002). Gambar 10.23. menunjukkan bagaimana hubungan *tabel/file/relation*, *row/record/tuple* dan *column/field/attribute*. Gambar ini juga menunjukkan susunan dari hubungan tersebut.



Table	Row	Column
File or Datafile	Record	Field
Relation	Tuple	Attribute



Gambar 10.23. Hubungan *tabel/file/relation*, *row/record/tuple* dan *column/field/attribute*.

*Relation schema* terdiri dari nama dari *relation*, nama dari *attribute* yang ada pada suatu *relation* beserta nama *domain*nya, dan *integrity constraints*. Nama dari *relation* haruslah unik dalam suatu basis data, atau tidak boleh ada nama *relation* yang sama. Nama *attribute* adalah nama kolom dari *relation* dan tidak ada nama *attribute* yang sama pada suatu *relation*. Nama domain dari suatu *attribute* berhubungan dengan tipe data yang digunakan oleh *attribute* tersebut. *Integrity constraints* adalah batasan pada *relational instances* pada suatu *schema* (Ramakrishnan and Gehrke, 2000; Lewis et al., 2002)

### 10.3.2. Struktur Basis Data Relasional

Seperti telah dijelaskan di atas sebuah tabel terdiri dari baris dan kolom.

- **Row/Baris/Tuple**

*Row/Baris/Tuple* adalah sekumpulan atribut yang saling berhubungan dalam satu baris (lihat Gambar 10.23). *Row* ini akan selalu berulang dengan struktur yang sama namun dengan isi data yang berbeda. Sebagai contoh, pada Gambar 10.23, row pertama memiliki struktur yang sama dengan *row* yang ke 2 dan ke 3. Namun data pada masing-masing *row* berbeda. Kita dapat mengidentifikasi untuk pembeli dengan *id\_pembeli* = 1 pasti memiliki nama = Cristiano Ronaldo dan untuk *id\_pembeli* = 2 pasti memiliki nama = Ryan Giggs, demikian seterusnya.

- **Field/Kolom/Attributes**

*Field/Kolom/Attributes* menunjukkan struktur dari data dari baris-baris yang berulang. Pada Gambar 10.23, terdapat 4 kolom, yaitu *id\_pembeli*, *nama*, *alamat*, dan *telepon*. Data pada kolom *id\_pembeli* misalnya, akan memiliki struktur yang sama, yaitu dalam bentuk angka dan merupakan urutan *id* pembeli. Demikian juga pada kolom *nama* yang hanya berisi nama pembeli saja, tidak bercampur dengan data lain.

Sebuah kolom harus memiliki nama kolom dan tipe data untuk data yang berada dalam kolom tersebut (Gambar 10.24). Selain itu, kadang-kadang juga ada pembatas (*constraint*) dan domain untuk data yang termasuk dalam kolom tersebut. Tipe data akan sangat bergantung pada atribut yang digunakan dan operasi-operasi yang akan dilakukan pada basis data ini. *Domain data* memiliki banyak kesamaan pengertian dengan fungsi tipe data yang digunakan. Namun, tipe data lebih merujuk pada kemampuan penyimpanan data yang mungkin bagi suatu atribut secara fisik, tanpa melihat layak tidaknya data tersebut bila dilihat dari pemakaian di dunia nyata. Sementara domain data lebih ditekankan pada batas-batas nilai yang diperbolehkan bagi suatu atribut, dilihat dari kenyataan pemakaiannya.

Name	Null?	Type
ISBN	NOT NULL	INTEGER
PUBLISHER_ID	NOT NULL	INTEGER
PUBLICATION_ID	NOT NULL	INTEGER
PRINT_DATE		DATE
PAGES		INTEGER
LIST_PRICE		INTEGER
FORMAT		VARCHAR (32)
RANK		INTEGER
INGRAM_UNITS		INTEGER

Diagram annotations:

- A callout box labeled "Column names" points to the "Name" column.
- A callout box labeled "NULL constraint" points to the "Null?" column.
- A callout box labeled "Datatype" points to the "Type" column.

Gambar 10.24. Kolom, constrain dan tipe data (Powell, 2006).

Pada Gambar 10.24 terlihat nama-nama kolom pada sebuah tabel lengkap dengan tipe data dan *constrain*nya. Kolom ISBN misalnya bertipe data *integer*, artinya kolom ini hanya boleh diisi dengan bilangan *integer*. Selain itu kolom ini memiliki *constrain no null*, yang berarti ketika mengisi data, kolom ini harus selalu terisi tidak boleh dikosongkan. Pada kolom *Pages*, tipe datanya adalah *date*, artinya hanya boleh berisi data berbentuk tanggal. Kolom *Pages* tidak memiliki *constrain null*, artinya kolom ini boleh diisi atau tidak diisi.

Jenis-jenis tipe data tergantung pada DBMS yang digunakan. Sebagai contoh MySQL memberikan jenis-jenis tipe data yang lebih luas dari pada Microsoft Access. Pada MySQL misalnya, kita akan menjumpai tipe data set, enum yang tidak dijumpai dalam Microsoft Access. Pemilihan tipe data yang tepat sangat penting karena mempengaruhi konsistensi data dan kinerja basis data.

Contoh domain adalah, apabila kita berhadapan dengan atribut / kolom tentang kelas pada Sekolah Dasar (SD). Kolom kelas ini hanya boleh diisi data angka 1 sampai dengan 6, karena tidak ada kelas 7 atau 4.5 di SD. Artinya domain kolom kelas adalah bilangan *integer* (bulat dan tidak ada pecahan) antara 1 sampai dengan 6 saja. Contoh lainnya adalah kolom nilai ujian, kolom ini domainnya adalah bilangan asli (real dan boleh pecahan) antara 0 sampai dengan 100.

Ada beberapa tipe atribut, yaitu:

- o Atribut sederhana (*Simple Attribute*), yaitu atribut *atomic* yang tidak dapat dipecah lagi.

kode_kursus	nama_kursus	lama_kursus
K0301	Pemrograman Web Dengan PHP	3
K0302	Dasar Pemrograman Web	1
K0303	Dasar MS Visual Basic	1
K0304	MS Visual Basic Lanjut	2
K0305	Basis Data Dengan MySQL	5

Atribut  
Sederhana

Gambar 10.25. Contoh atribut sederhana

- o Atribut komposit (*Composit Attribute*), yaitu atribut yang masih dapat dipecah lagi menjadi sub-sub atribut yang masing-masing memiliki makna.

kode_siswa	nama_siswa	alamat_siswa	tgl_lahir_siswa
S090001	Cristiano Ronaldo	Jl. Manchester Raya No.10 Manchester 65144	22-Nov-90
S090002	Ryan Giggs	Jl. Simpang Manchester II/10 Manchester 65123	12-Jul-91
S090003	Wayne Rooney	Jl. Simpang Manchester III/1 Manchester 65132	02-Apr-90
S090004	Cesc Fabregas	Jl. Raya Camp Nou 13 Barcelona 54003	30-Jun-93
S090005	Robbie van Persie	Jl. Simpang Ponderland I/45 Eindhoven 47222	25-Sep-89

Atribut alamat\_siswa merupakan atribut komposit yang dapat dipecah menjadi tiga sub atribut

alamat	kota	kode_pos
Jl. Manchester Raya No.10	Manchester	65144
Jl. Simpang Manchester II/10	Manchester	65123
Jl. Simpang Manchester III/1	Manchester	65132
Jl. Raya Camp Nou 13	Barcelona	54003
Jl. Simpang Ponderland I/45	Eindhoven	47222

Gambar 10.26. Contoh atribut komposit.

- o Atribut bernilai tunggal (*Single-Valued Attribute*), yaitu atribut yang memiliki paling banyak satu nilai untuk setiap baris data.
- o Atribut bernilai banyak (*Multi-Valued Attribute*), yaitu atribut yang dapat berisi lebih dari satu nilai tetapi dengan jenis yang sama.

kode_instruktur	nama_instruktur	alamat_instruktur	keahlian_instruktur
I04001	Felipe Scolari	Jl. Terusan Sao Paulo III/67 Rio de Janeiro	Sistem Basis Data Pemrograman PHP
I04002	Sven Goran Eriksson	Jl. Upsalla Raya 23 Gothenborg	Pemrograman Visual
I04003	Steve Mc Laren	Jl. Merseyside Utara 12 Liverpool	Pemrograman Terstruktur Pemrograman Shell

Atribut bernilai tunggal

Atribut bernilai banyak

Gambar 10.27. Contoh atribut bernilai tunggal dan atribut bernilai banyak.

- o Atribut turunan (*Derived Attribute*), yaitu atribut yang nilai-nilainya diperoleh dari hasil pengolahan atau dapat diturunkan dari atribut atau tabel lain yang berhubungan.

### 10.3.3. Relasi Antar Tabel

Keistimewaan utama basis data relasional dibandingkan model basis data lainnya adalah kemudahan dalam membangun hubungan antar tabel dalam bentuk yang masuk akal dapat dimengerti. Relasi antar tabel dapat kita turunkan langsung ataupun tak langsung dari ER-Diagram yang telah dibahas sebelumnya. Entitas yang ada pada ER-Diagram biasanya merupakan kandidat dari suatu tabel pada basis data relasional. Relasi antar tabel biasanya dapat diidentifikasi dari relationship antar entitas pada ER-Diagram.

Untuk memperjelas bagaimana basis data relasional menangani hubungan antar tabel kita akan menggunakan tabel-tabel berikut ini. Tabel pertama adalah Pengarang. Tabel ini terdiri dari 3 kolom yaitu id\_pengarang, pengarang dan tahun\_lahir (Gambar 10.28). Pada tabel ini id\_pengarang merupakan *primary key*.

Pengarang : Table		
id_pengarang	pengarang	tahun_lahir
1	Adams, Pat	
2	Adrian, Merv	
3	Ageloff, Roy	1943
4	Andersen, Virginia	
5	Antonovich Michael P.	
6	Arnott, Steven E.	
7	Amtson, L. Joyce	
8	Ault, Michael R	
9	Avison, D. E.	
10	Bard, Dick	1941
11	Biegel, Richard A	
12	Blow, Lisa.	
13	Bisland, Ralph B	
14	Bowman, Judith S	

Gambar 10.28. Tabel Pengarang.

Tabel kedua adalah Penerbit. Tabel ini mempunyai 6 kolom yaitu, id\_penerbit, nama, nama\_perusahaan, alamat, kota dan telepon (Gambar 10.29). *Primary key* pada tabel ini adalah id\_penerbit.

Penerbit : Table					
id_penerbit	nama	nama_perusahaan	alamat	kota	telepon
1	ACM	Association for Computing Machinery	11 W. 42nd St., 3rd flr.	New York	212-869-7440
2	Addison-Wesley	Addison-Wesley Publishing Co Inc.	Rte 128	Reading	617-944-3700
3	Bantam Books	Bantam Books Div of Bantam Doubleday De	666 Fifth Ave	New York	800-223-6834
4	Benjamin/Cummings	Benjamin-Cummings Publishing Company S	390 Bridge Pkwy.	Redwood City	800-950-2665
5	Brady Pub	Brady Books Div. of Prentice Hall Pr., Simor	15 Columbus Cir.	New York	212-373-8093
6	Computer Science Press	Computer Science Press Inc Imprint of W H	41 Madison Ave	New York	212-576-9400
7	ETN Corporation	ETN Corp.	RD 4, Box 659	Montoursville	717-435-2202
8	Gale	Gale Research, Incorporated	835 Penobscot Bldg	Detroit	313-961-2242
9	IEEE	IEEE Computer Society Press	10662 Los Vaqueros Circle	Los Alamitos	800-272-6657
10	Intertext	Intertext Publications/Multiscience Press	2633 E. 17th Ave.	Anchorage	
11	M&T Books	M & T Books Div of M&T Publishing Inc	501 Galveston Dr	Redwood City	800-533-4372
12	Macmillan Education	Macmillan Education Ltd	175 Fifth Ave	New York	212-460-1500
13	McGraw-Hill	McGraw-Hill Inc	1221 Ave of the Americas	New York	212-512-2000
14	Microsoft Press	Microsoft Press Div of Microsoft Corp	One Microsoft Way	Redmond	800-MSPRESS
15	Morgan Kaufmann	Morgan Kaufmann Publishers Inc.	2929 Campus Dr, Suite 260	San Mateo	415-578-9911

Gambar 10.29. Tabel Penerbit.

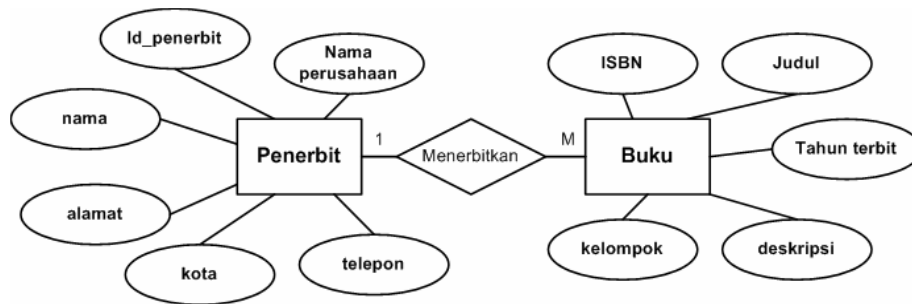
Tabel ketiga adalah Buku. Tabel ini mempunyai 6 kolom yaitu, judul, tahun\_terbit, ISBN, id\_penerbit, deskripsi, dan kelompok (Gambar 10.30). *Primary key* pada tabel ini adalah ISBN. Ada yang sedikit aneh pada tabel ini, yaitu kolom id\_penerbit yang merupakan salah satu kolom pada tabel Penerbit, dimasukkan dalam tabel ini. Sebenarnya ini bukan keanehan atau kesalahan, tetapi memang beginilah salah satu cara basis data relasional menangani hubungan antar tabel.

Buku : Table					
judul	tahun_terbit	ISBN	id_penerbit	deskripsi	kelompok
Database management: developing application systems using	1989	0-0131985-2-1	17	xx, 441 p. ; ill. ;	
Select-- SQL : the relational database language	1992	0-0238669-4-2	12	xv, 446 p. ; 24 c	
dBase IV programming	1994	0-0280042-4-8	73		
Step-by-step dBase IV	1995	0-0280095-2-5	52		
Guide to ORACLE	1990	0-0702063-1-7	13	xii, 354 p. ; ill. ;	ORACLE (Computer syste
The database experts' guide to SQL	1988	0-0703900-6-1	10		
Oracle/SQL: a professional programmer's guide	1992	0-0704077-5-4	13	xx, 543 p. ; ill. ;	
SQL 400: A Professional Programmer's Guide	1994	0-0704079-9-1	52		
Database system concepts	1986	0-0704475-2-7	13		
Microsoft FoxPro 2.5 applications programming	1993	0-0705015-3-X	61	xiii, 412 p. ; ill. ;	
First look at-- dBASE IV, version 1.5/2.0 for DOS	1994	0-0705107-5-X	80	ill. ; 24 cm.	
Applying SQL in Business	1992	0-0705184-2-4	13		
Database design	1977	0-0707013-0-X	13		
Introduction to Oracle	1989	0-0770716-4-6	13	xi, 342 p. ; 24 c	

Gambar 10.30 Tabel buku.

Dalam dunia nyata, kita akan menjumpai bahwa satu penerbit tidak hanya menghasilkan satu judul buku saja, tetapi ratusan bahkan mungkin jutaan judul buku. Sehingga secara formal hubungan antara penerbit dengan buku dapat dinyatakan sebagai hubungan dengan kardinalitas *one-to-many*. Jika digambarkan dalam bentuk ER-Diagram akan tampak seperti Gambar 10.31. Satu penerbit dapat menerbitkan banyak judul buku dan satu judul buku hanya diterbitkan oleh satu penerbit. Untuk lebih memperjelas perhatikan Gambar

10.31. Penerbit dengan id\_penerbit = 13 (McGraw Hill) menerbitkan 6 buah buku. Sebaliknya buku dengan ISBN = 0-0702063-1-7 (Guide To Oracle) hanya diterbitkan oleh penerbit dengan id\_penerbit = 13 (McGraw Hill). Sehingga penempatan kolom id\_penerbit pada tabel buku dimaksudkan untuk merepresentasikan hubungan Penerbit dengan Buku. Kolom id\_penerbit pada tabel buku biasa disebut sebagai *foreign key*.



Gambar 10.31. ER-Diagram untuk Penerbit dan Buku

Penerbit : Table					
id_penerbit	nama	nama_perusahaan	alamat	kota	telepon
1	ACM	Association for Computing Machinery	11 W. 42nd St., 3rd flr.	New York	212-869-7440
2	Addison-Wesley	Addison-Wesley Publishing Co Inc.	Rte 128	Reading	617-944-3700
3	Bantam Books	Bantam Books Div of. Bantam Doubleday De	666 Fifth Ave	New York	800-223-6934
4	Benjamin/Cummings	Benjamin-Cummings Publishing Company S	390 Bridge Pkwy.	Redwood City	800-950-2665
5	Brady Pub.	Brady Books Div. of Prentice Hall Pr., Simor	15 Columbus Cir.	New York	212-373-8093
6	Computer Science Press	Computer Science Press Inc Imprint of W H	41 Madison Ave	New York	212-576-9400
7	ETN Corporation	ETN Corp.	RD 4, Box 659	Montoursville	717-435-2202
8	Gale	Gale Research, Incorporated	835 Penobscot Bldg	Detroit	313-961-2242
9	IEEE	IEEE Computer Society Press	10662 Los Vaqueros Circle	Los Alamitos	800-272-6657
10	Intertext	Intertext Publications/Multiscience Press	2633 E. 17th Ave.	Anchorage	
11	M&T Books	M & T Books Div of. M&T Publishing Inc	501 Galveston Dr	Redwood City	800-533-4372
12	Macmillan Education	Macmillan Education Ltd	175 Fifth Ave	New York	212-460-1500
13	McGraw-Hill	McGraw-Hill Inc	1221 Ave of the Americas	New York	212-512-2000
14	Microsoft Press	Microsoft Press Div of. Microsoft Corp	One Microsoft Way	Redmond	800-MSPRESS
15	Morgan Kaufmann	Morgan Kaufmann Publishers Inc.	2929 Campus Dr. Suite 260	San Mateo	415-578-9911

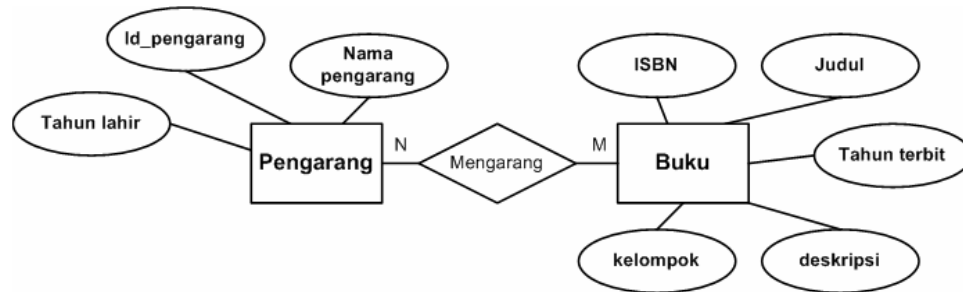
Buku : Table					
judul	tahun_terbit	ISBN	id_penerbit	deskripsi	kelompok
Database management: developing application systems using	1989	0-0131985-2-1	17	xx, 441 p. : ill. :	
Select-- SQL : the relational database language	1992	0-0238669-4-2	12	xv, 446 p. : 24 c	
dBase IV programming	1994	0-0280042-4-8	73		
Step-by-step dBase IV	1995	0-0280095-2-5	52		
Guide to ORACLE	1990	0-0702063-1-7	13	354 p. : ill. : ORACLE (Computer systems)	
The database experts' guide to SQL	1988	0-0703900-6-1	10		
Oracle/SQL: a professional programmer's guide	1992	0-0704077-5-4	13	543 p. : ill. :	
SQL 400: A Professional Programmer's Guide	1994	0-0704079-9-1	52		
Database system concepts	1986	0-0704475-2-7	13		
Microsoft FoxPro 2.5 applications programming	1993	0-0705015-3-X	61	xiii, 412 p. : ill. :	
First look at-- dBASE IV, version 1.5/2.0 for DOS	1994	0-0705107-5-X	80	ill. : 24 cm.	
Applying SQL in Business	1992	0-0705184-2-4	13		
Database design	1977	0-0707013-0-X	13		
Introduction to Oracle	1989	0-0770716-4-6	13	xi, 342 p. : 24 cm	

Gambar 10.32. Hubungan tabel Penerbit dan Buku.

Pada tabel-tabel yang telah dibuat di atas kita belum melihat tabel yang menunjukkan seorang pengarang tertentu mengarang buku apa. Untuk



membuat tabel yang berisi pengarang dan buku karangannya, kita dapat menggunakan hubungan antara tabel pengarang dengan tabel buku. Namun sebelum itu kita harus melihat bagaimana hubungan antara pengarang dengan buku di dunia nyata. Seorang pengarang mungkin hanya mengarang satu judul buku, tetapi mungkin juga lebih. Sedangkan satu buku, mungkin ditulis hanya oleh satu orang pengarang, tetapi mungkin juga ditulis oleh dua, tiga atau lebih pengarang. Sehingga kita bisa mengatakan kardinalitas hubungan pengarang dengan buku adalah many-to-many. Kita dapat menggambarkan ER-Diagram untuk kasus ini seperti pada gambar 10.33.



Gambar 10.33. ER-Diagram untuk Pengarang – Buku.

Pada kasus dengan kardinalitas many-to-many kita tidak dapat langsung menyisipkan satu foreign key pada tabel lain. Kita harus membuat tabel baru agar kardinalitas antar tabel yang terlibat dapat diubah menjadi one-to-many. Tabel Pengarang\_Buku merupakan tabel yang dibentuk untuk menangani hubungan tabel Buku dengan tabel Pengarang. Tabel ini hanya berisi dua atribut (kolom) yaitu ISBN yang berasal dari tabel Buku dan id\_pengarang yang berasal dari tabel Pengarang. Pada Gambar 10.34, terlihat pada tabel Pengarang\_Buku ada beberapa buku yang dikarang lebih dari satu pengarang.



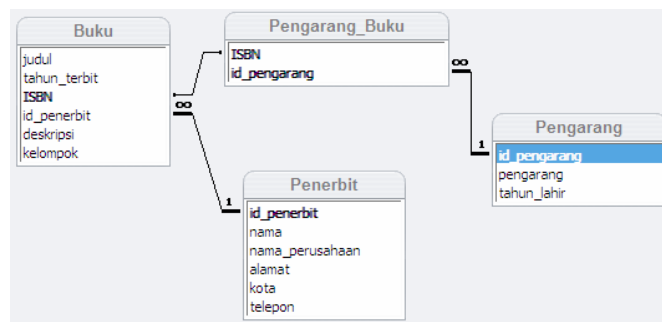
Buku : Table					
judul	tahun_terbit	ISBN	id_penerbit	deskripsi	kelompok
Database management: developing application systems using	1989	0-0131985-2-1	17	xx, 441 p. : ill. :	
Select-- SQL : the relational database language	1992	0-0238669-4-2	12	xv, 446 p. : 24 c	
dBase IV programming	1994	0-0280042-4-8	73		
Step-by-step dBase IV	1995	0-0280095-2-5	52		
Guide to ORACLE	1990	0-0702063-1-7	13	xii, 354 p. : ill. :	ORACLE (Computer syste
The database experts' guide to SQL	1988	0-0703900-6-1	10		
Oracle/SQL: a professional programmer's guide	1992	0-0704077-5-4	13	xx, 543 p. : ill. :	
SQL 400: A Professional Programmer's Guide	1994	0-0704079-9-1	52		
Database system concepts	1986	0-0704475-2-7	13		
Microsoft FoxPro 2.5 applications programming	1993	0-0705015-3-X	61	xi	
First look at-- dBASE IV, version 1.5/2.0 for DOS	1994	0-0705107-5-X	80	ill	
Applying SQL in Business	1992	0-0705184-2-4	13		
Database design	1977	0-0707013-0-X	13		
Introduction to Oracle	1989	0-0770716-4-6	13	xi	

Pengarang : Table		
id_pengarang	pengarang	tahun_lahir
1	Adams, Pat	
2	Adrian, Merv	
3	Ageloff, Roy	1943
4	Andersen, Virginia	
5	Antonovich, Michael P.	
6	Amott, Steven E.	
7	Amtson, L. Joyce	
8	Ault, Michael R.	
9	Avison, D. E.	
10	Bard, Dick	1941
11	Biegel, Richard A.	
12	Blow, Lisa	
13	Bisland, Ralph B.	
14	Bowman, Judith S.	

Pengarang_Buku : Table	
ISBN	id_pengarang
0-0131985-2-1	13
0-0238669-4-2	113
0-0280042-4-8	11
0-0280042-4-8	120
0-0280095-2-5	171
0-0702063-1-7	26
0-0702063-1-7	65
0-0702063-1-7	104
0-0703900-6-1	96
0-0704077-5-4	59
0-0704077-5-4	99
0-0704079-9-1	59

Buku yang dikarang lebih dari satu pengarang

Gambar 10.34. Hubungan tabel Pengarang dan Buku.



Gambar 10.35. Relasi antar tabel.

#### 10.4. RINGKASAN

- Basis data (database) merupakan kumpulan dari data yang saling berhubungan satu dengan yang lainnya, tersimpan dalam perangkat keras komputer dan digunakan perangkat lunak untuk memanipulasinya.
- Operasi dasar basis data meliputi pembuatan basis data baru, penghapusan basis data, pembuatan tabel baru, penghapusan tabel,

pengisian atau penambahan data baru, pengambilan data, perubahan data, penghapusan data.

- Sistem Manajemen Basis Data merupakan perangkat lunak yang bekerja khusus untuk menangani basis data.
- Entity-Relationship Diagram merupakan notasi grafis dalam pemodelan data konseptual yang digunakan untuk memodelkan struktur data dan hubungan antar data. Elemen dari ERD adalah entitas, atribut, relationship dan kardinalitas.
- Pembentuk utama dalam model data relasional adalah *relation* (tabel). *Relation instance* adalah sebuah tabel dua dimensi dengan baris (row/record/tuple) dan kolom (column/field/attribute).
- Pada basis data relasional, identifikasi yang tepat pada hubungan antar atribut di dalam satu tabel dan hubungan antar tabel merupakan kunci membuat basis data yang baik.

#### 10.5. SOAL-SOAL LATIHAN

1. Berkunjuglah ke perpustakaan sekolah, kemudian buatlah pengamatan singkat. Buatlah catatan untuk menentukan siapa dan apa yang terlibat dalam kegiatan perpustakaan sekolah. Cermatilah mana yang bisa digolongkan sebagai entitas, atribut. Dan bagaimana hubungan antar entitas.
2. Dari hasil kegiatan no. 1 kemudian buatlah tabel-tabel yang menunjukkan entitas dan hubungannya. Tentukan pula atribut dari masing-masing tabel.
3. Cermati pula tipe-tipe atribut yang telah kalian tentukan.

- Anonymous. 2004. Guide to the Software Engineering Body of Knowledge (SWEBOK). The Institute of Electrical and Electronics Engineers, Inc.
- Balter, A. 2006. Sams Teach Yourself Microsoft® SQL Server™ 2005 Express in 24 Hours. Sams.
- Bass, L., P. Clements, and R. Kazman. 2003. Software Architecture in Practice. 2<sup>nd</sup> Edition. Addison-Wesley.
- Cormen, T.H. 2001. Introduction to Algorithm: Second Edition. The MIT Press.
- Deek, F.P., J.A.M. McHugh, and O.M. Eljabiri. 2005. Strategic software engineering : An Interdisciplinary Approach. Auerbach Publications.
- den Haan, P., L. Lavandowska, S.N. Panduranga, and K. Perrumal. 2004. Beginning JSP 2: From Novice to Professional. Apress.
- Dobson, R. 1999. Programming Microsoft Access 2000: The Developer's Guide to Harnessing the Power of Access. Microsoft Press.
- Felleisen, M, R.B. Findler, M. Flatt, and S. Krishnamurthi. 2001. How to Design Programs; An Introduction to Computing and Programming. The MIT Press.
- Kak, A.C. 2003. Programming With Objects: A Comparative Presentation of Object Oriented Programming with C++ and Java. John Wiley & Sons, Inc.
- Kaisler, S.H. 2005. Software Paradigm. John Wiley & Sons, Inc.
- Kennedy, B. and C. Musciano. 2006. HTML & XHTML: The Definitive Guide, 6th Edition. O'Reilly.
- Lafore, R. 1998. Data Structures & Algorithm in Java. Waite Group Press.
- Laurie, B and P. Laurie. 2001. Apache: The Definition Guide. 2<sup>nd</sup> Edition. O'Reilly and Associates, Inc.
- Leffingwell, D. and D. Widrig. 2003. Managing Software Requirements: A Use Case Approach. 2<sup>nd</sup> Edition. Addison-Wesley.
- Lischner, R. 2000. Delphi in a Nutshell. O'Reilly and Associates, Inc.

- Luckey, T. and J. Phillips. 2006. Software Project Management for Dummies. Wiley Publishing, Inc.
- McConnel, S. 2003. Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers. Addison-Wesley.
- Meyer, B. 2000. Object Oriented Software Construction. 2<sup>nd</sup> Edition. ISE, Inc.
- Musciano, C. and B. Kennedy. 2002. HTML and XHTML: The Definition Guide. 4<sup>th</sup> Edition. O'Reilly and Associates, Inc.
- Navarro, A. 2001. Effective Web Design. 2<sup>nd</sup> Edition. SYBEX, Inc.
- Powell, G. 2006. Beginning Database Design. Wiley Publishing, Inc.
- Riordan, R.M. 2005. Designing Effective Database Systems. Addison Wesley Professional.
- Robbins, J. N. 2006. Web Design in a Nutshell, 3rd Edition. O'Reilly.
- Suehring, S. 2002. MySQL Bible. Wiley Publishing, Inc.
- Taylor, D.A. 1998. Object Technology: A Manager's Guide. Addison-Wesley.
- Van Roy, P and S. Haridi. 2004. Concepts, Techniques, and Models of Computer Programming. The MIT Press.

---

# Lampiran 1

## Daftar Istilah / Glosari

---

**Abstraction**

Merupakan prinsip penyederhanaan dari sesuatu yang kompleks dengan cara memodelkan kelas sesuai dengan masalahnya

**Algoritma**

Urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis

**Array**

Struktur data yang menyimpan sekumpulan elemen yang bertipe sama

**Atribut**

Karakteristik atau ciri yang membedakan antara entitas satu dengan entitas yang lainnya

**Authentication**

Proses memeriksa keabsahan seseorang sebagai user (pengguna) pada suatu system (misalnya pada DBMS)

**Basic Input/Output System (BIOS)**

Kode-kode program yang pertama kali dijalankan ketika komputer dinyalakan (booting)

**Basis data (database)**

Kumpulan dari data yang saling berhubungan satu dengan yang lainnya, tersimpan dalam perangkat keras komputer dan digunakan perangkat lunak untuk memanipulasinya

**Command Line Interface (CLI)**

Antar muka pengguna dengan model perintah-perintah teks

**Compiler**

Penerjemah bahasa pemrograman tingkat tinggi ke bahasa mesin dengan cara sekaligus seluruh kode program. Prosesnya disebut kompilasi.

**Component Object Model (COM)**

Infrastruktur yang disediakan oleh Visual Basic untuk mengakses obyek-obyek atau kontrol-kontrol lain sepanjang punya antar muka yang dapat diakses oleh Visual Basic.

**Constraint**

Batasan-batasan dari masalah

**Control**

Aktivitas monitoring dan evaluasi terhadap feedback untuk menentukan apakah system telah bekerja dengan baik atau tidak

**Counter**

Variable pencacah yang digunakan dalam struktur algoritma pengulangan

**Database Management System (DBMS)**

Perangkat Lunak yang khusus / spesifik ditujukan untuk pengelolaan basis data

**Disk Operating System (DOS)**

Salah satu sistem operasi lama berbasis CLI

**Elektronika**

Ilmu yang mempelajari alat listrik **arus lemah** yang dioperasikan dengan cara mengontrol aliran elektron atau partikel bermuatan listrik dalam suatu alat

**Entitas**

Individu yang mewakili sesuatu yang nyata (eksistensinya) dan dapat dibedakan dari sesuatu yang lain

**Extensible Hypertext Markup Language (XHTML)**

HTML versi terakhir (4.01) yang ditulis ulang dengan aturan-aturan yang lebih ketat mengacu pada XML

**Extensible Markup Language (XML)**

Sekumpulan aturan untuk menyusun bahasa *markup*

**Feedback**

Data tentang kinerja sistem

**Flowchart**

Skema/bagan (*chart*) yang menunjukkan aliran (*flow*) di dalam suatu program secara logika

**Gejala**

Signal atau tanda terjadinya suatu masalah

**Gerbang logika**

blok-blok penyusun dari perangkat keras elektronik

**Graphical User Interface (GUI)**

Antar muka pengguna dengan model grafis

**Identifier**

Nama dari suatu variable atau konstanta

**Ilmu komputer**

Suatu studi sistematis pada proses-proses algoritma yang menjelaskan dan mentransfor-masikan informasi

**Inheritance atau pewarisan**

Prinsip pewarisan sifat dari orang tua ke anak atau turunannya yang diterapkan pada kelas

**Inisialisasi**

Instruksi yang dilakukan pertama kali pada suatu variabel atau ekpresi pemrograman

**Input**

Elemen-elemen yang masuk ke dalam system

**Integrated Developement Environment (IDE)**

Lingkungan pengembangan aplikasi terintegrasi. Perangkat lunak untuk membantu mempermudah pembuatan aplikasi komputer

**Interpreter**

Penerjemah bahasa pemrograman tingkat tinggi ke bahasa mesin dengan cara satu per satu baris dibaca dan langsung diterjemahkan

**Kardinalitas**

Jumlah maksimum entitas pada suatu himpunan entitas yang dapat berelasi dengan entitas pada himpunan entitas yang lain

**Konstanta**

Variabel yang nilai datanya bersifat tetap dan tidak bisa diubah.

**Loop**

Proses pengulangan suatu perintah

**Masalah (problem)**

Perbedaan antara situasi aktual dan situasi yang diharapkan atau perbedaan antara kondisi sekarang dengan target atau tujuan yang diinginkan

**Model**

Penyederhanaan dari suatu system atau Tiruan dari suatu sistem dengan sedikit atau banyak penyederhanaan

**Multi-tasking**

Kemampuan sistem operasi untuk menjalankan beberapa tugas / aplikasi secara bersamaan

**Multi-user**

Kemampuan system operasi untuk dijalankan oleh pengguna yang berbeda pada waktu bersamaan

**Output**

Perpindahan elemen-elemen yang dihasilkan dari proses perubahan ke tujuan yang diinginkan

**Pemecahan masalah**

Sebuah proses dimana suatu situasi dianalisa kemudian solusi-solusi dibuat bila ditemukan ada masalah dengan cara pendefinisian, pengurangan atau penghilangan, atau pencegahan masalah

**Pemrograman Berorientasi Obyek (*Object Oriented Programming – OOP*)**

Paradigma pemrograman yang menggunakan obyek dan interaksinya untuk merancang aplikasi dan program komputer

**Pemrograman web**

Usaha untuk membuat halaman web dengan menggunakan bahasa pemrograman web (*script*)

**Perangkat lunak**

Seluruh instruksi yang digunakan untuk memproses informasi

**Permissions**

Proses untuk menentukan apa yang bisa dilakukan seorang pengguna pada suatu sistem

**Pointer**

Variabel yang menyimpan alamat pada memori komputer

**Polymorphism**

Kemampuan dari suatu obyek untuk mempunyai lebih dari satu bentuk

**Programmer**

Seseorang yang bekerja membuat program komputer

**Prosedur**

- Instruksi yang dibutuhkan oleh pengguna dalam memproses informasi
- Sekumpulan perintah yang merupakan bagian dari program yang lebih besar yang berfungsi mengerjakan suatu tugas tertentu

**Proses**

Perubahan atau transformasi input menjadi output



**Prototyping**

Salah satu pendekatan dalam pengembangan perangkat lunak yang secara langsung mendemonstrasikan bagaimana sebuah perangkat lunak atau komponen-komponen perangkat lunak akan bekerja dalam lingkungannya sebelum tahapan konstruksi aktual dilakukan

**Pseudocode**

Cara penulisan algoritma dengan menggunakan kode-kode yang mirip dengan bahasa pemrograman

**Query**

Permintaan atau pencarian pada data-data tertentu pada suatu basis data

**Record**

Baris data dari suatu tabel

**Rekayasa Perangkat Lunak**

suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu analisa kebutuhan pengguna, menentukan spesifikasi dari kebutuhan pengguna, disain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan

**Relationship atau relasi**

Hubungan yang terjadi antara sejumlah entitas

**Sistem**

Kumpulan dari elemen-elemen yang saling berinteraksi untuk mencapai tujuan tertentu

**Sistem basis data**

Kumpulan elemen-elemen seperti basis data, perangkat lunak, perangkat keras, dan manusia yang saling berinteraksi untuk mencapai tujuan yaitu pengorganisasian data.

**Software**

Lihat Perangkat Lunak

**Software Engineering**

Lihat Rekayasa Perangkat Lunak

**Solusi**

Bagian akhir atau output dari proses pemecahan masalah.

**Stored procedure**

Potongan kode program yang dapat menerima parameter input dan menghasilkan satu atau lebih parameter output dan digunakan untuk operasi-operasi basis data

### **Structured Query Language (SQL)**

Bahasa query terstruktur untuk mengelola basis data

### **Strategi pemecahan masalah**

Metode atau pendekatan yang digunakan seseorang ketika menghadapi masalah

### **Struktur algoritma**

Cara atau urutan untuk membuat suatu algoritma

### **Tipe data**

Jenis data yang dapat diolah oleh komputer untuk memenuhi kebutuhan dalam pemrograman komputer

### **Trigger**

Tipe khusus dari stored procedure yang akan dieksekusi ketika suatu kejadian muncul

### **Variabel**

Tempat dimana kita dapat mengisi atau mengosongkan nilainya dan memanggil kembali apabila dibutuhkan pada suatu program

### **View**

Tabel virtual yang isinya berdasarkan pada query yang dilakukan pada basis data.

### **Web browser**

Perangkat lunak yang berfungsi menerjemahkan kode-kode HTML menjadi tampilan yang kita kehendaki

### **Web dinamis**

Halaman-halaman web yang isi dan informasinya berubah-ubah sesuai dengan permintaan pengguna

### **Web server**

Perangkat lunak yang bertindak melayani permintaan-permintaan *client* terhadap halaman-halaman *web* tertentu

### **Web statis**

Halaman-halaman web yang isi dan informasinya tidak berubah-ubah

---

## Lampiran 2

# Daftar Alamat Situs

---

Berikut ini daftar alamat situs-situs internet yang penting dan digunakan sebagai rujukan dalam buku ini.

Alamat	Keterangan
<a href="http://www.apache.org">http://www.apache.org</a>	Situs resmi web server Apache. Situs ini menyediakan kode sumber Apache dan file-file binary Apache yang siap diinstall di berbagai platform sistem operasi. Selain itu juga menyediakan dokumentasi Apache yang lengkap.
<a href="http://www.borland.com">http://www.borland.com</a>	Situs resmi Borland. Borland merupakan perusahaan perangkat lunak yang memproduksi Borland Delphi, Borland JBuilder, Turbo Pascal, Turbo Delphi, Borland C++ dan lain-lain.
<a href="http://www.debian.org">http://www.debian.org</a>	Situs resmi distribusi linux Debian.
<a href="http://www.eclipse.org">http://www.eclipse.org</a>	Situs resmi proyek eclipse, perangkat pengembang terpadu yang mendukung banyak bahasa pemrograman.
<a href="http://www.google.com">http://www.google.com</a>	Situs resmi search engine Google.
<a href="http://www.ilmukomputer.com">http://www.ilmukomputer.com</a>	Situs berbahasa Indonesia yang menyediakan dokumen-dokumen untuk belajar berbagai sub bidang dalam ilmu computer.
<a href="http://www.javasoft.com">http://www.javasoft.com</a>	Situs resmi yang diluncurkan Sun Microsystem dan berisi dokumentasi dan informasi online tentang bahasa pemrograman Java.
<a href="http://www.kambing.vlsm.org">http://www.kambing.vlsm.org</a>	Situs dengan server local di Indonesia. Situs ini menyediakan file-file iso dari berbagai jenis distribusi linux dan dapat didownload secara bebas. Selain itu situs ini juga sebagai mirror dari berbagai distribusi linux dan aplikasi yang berjalan di linux.
<a href="http://www.linuxdoc.org">http://www.linuxdoc.org</a>	Situs yang berisi dokumentasi bebas tentang linux. Sumber informasi online yang sangat bagus untuk mempelajari linux

<a href="http://www.microsoft.com">http://www.microsoft.com</a>	Situs resmi Microsoft. Microsoft merupakan perusahaan perangkat lunak yang memproduksi system operasi keluarga Windows, IDE Microsoft Visual Studio, Microsoft Office, Microsoft SQL Server, dan lain-lain.
<a href="http://www.mysql.com">http://www.mysql.com</a>	Situs resmi MySQL Database Software. Situs ini menyediakan file-file instalasi MySQL untuk berbagai platform sistem operasi. Selain itu juga menyediakan dokumentasi MySQL yang lengkap.
<a href="http://www.netbeans.org">http://www.netbeans.org</a>	Situs resmi IDE Netbeans, perangkat lunak pengembang aplikasi Java
<a href="http://www.php.net">http://www.php.net</a>	Situs resmi bahasa pemrograman dan interpreter PHP. Situs ini menyediakan kode sumber dan file-file instalasi PHP untuk berbagai platform sistem operasi. Selain itu juga menyediakan dokumentasi PHP yang lengkap.
<a href="http://www.w3.org">http://www.w3.org</a>	Situs resmi The World Wide Web Consortium (W3C). W3C adalah konsorsium yang menetapkan standar dalam teknologi internet, terutama tentang HTML, XML, CSS, XHTML dan teknologi lain. Dokumentasi tentang teknologi tersebut dapat dijumpai di situs ini.

---

## Lampiran 3

# Fungsi-fungsi Built-in Pada Visual Basic

---

### **IsNumeric(ekspresi)**

Fungsi ini digunakan untuk menguji apakah suatu ekspresi menghasilkan nilai numeric atau bukan. Nilai yang dikembalikan adalah Boolean.

### **IsEmpty(ekspresi)**

Fungsi untuk memeriksa apakah suatu ekspresi telah berisi nilai atau tidak. Nilai yang dikembalikan adalah Boolean..

### **IsNull(ekspresi)**

Fungsi untuk memeriksa apakah suatu ekspresi mengandung data yang tidak valid, biasanya digunakan untuk memeriksa isi field recordset.

### **IsArray(varname)**

Fungsi untuk memeriksa apakah suatu variabel adalah suatu array.

### **IsDate(ekspresi)**

Fungsi untuk memeriksa apakah suatu ekspresi dapat dikonversi ke date.

### **IsError(ekspresi)**

Fungsi untuk memeriksa apakah suatu ekspresi adalah nilai error

### **IsObject(ekspresi)**

Fungsi untuk memeriksa apakah suatu ekspresi mengacu pada suatu OLE Automation object.

### **IsMissing(argname)**

Fungsi untuk memeriksa apakah suatu argumen optional pada procedure ada dilewatkan atau tidak

### **CBool(ekspresi)**

Konversi suatu ekspresi ke Boolean

### **CByte(ekspresi)**

Konversi ekspresi ke Byte

### **CCur(ekspresi)**

Konversi suatu ekspresi ke Currency

### **CDate(date)**

Konversi suatu ekspresi ke date

### **Cdbl(ekspresi)**

Konversi suatu ekspresi ke Double

### **CInt(ekspresi)**

Konversi suatu ekspresi ke Integer

### **CLng(ekspresi)**

Konversi suatu ekspresi ke Long

**CSng(ekspresi)**

Konversi suatu ekspresi ke single

**CStr(ekspresi)**

Konversi suatu ekspresi ke string

**CVar(ekspresi)**

Konversi suatu ekspresi ke Variant

**Asc(string)**

Fungsi untuk menampilkan kode character dari huruf pertama di suatu string.

**Chr(charcode)**

Fungsi untuk menampilkan karakter dari suatu kode karakter

**Format(ekspresi[, format[, hariPertamaDariMinggu[, mingguPertamaDariTahun]])**

Memformat suatu ekspresi berdasarkan ekspresi format

**Hex(number) dan Oct(number)**

Menampilkan string yang mewakili Octal atau Hexa dari suatu bilangan

**Str(number)**

Menampilkan string yang mewakili suatu angka.

**Val(string)**

Menampilkan angka yang terkandung dalam suatu string.

**Now**

Mengembalikan suatu Variant (Date) yang menunjukkan tanggal dan waktu berdasarkan sistem komputer.

**Time**

Mengembalikan waktu sistem sekarang

**Timer**

Mengembalikan suatu bilangan yang menunjukan jumlah detik sejak tengah malam

**Date**

Mengembalikan tanggal sistem sekarang

**Time = Time dan Date = Date**

Mengatur waktu atau tanggal sistem

Untuk sistem yang menjalankan Microsoft Windows 95, tanggal yang dibutuhkan harus berupa tanggal dari 1 Jan 1998 sampai 31 Des 2099. Untuk sistem yang menjalankan Microsoft Windows NT, tanggal yang dibutuhkan harus berupa tanggal dari 1 Jan 1980 sampai 31 Desember 2079.

**Hour(time), Minute(time) dan Second(time)**

Mengembalikan suatu Variant (Integer) berupa bilangan 0 s/d 23 untuk jam, 0 s/d 59 untuk menit, dan 0 s/d 59 untuk detik.

**Day(date), Month(date), dan Year(date)**

Mengembalikan suatu Variant (Integer) berupa bilangan 1 s/d 31 untuk bulan, 1 s/d 12 untuk bulan, dan tahun.



ISBN 978-979-060-007-2

ISBN 978-979-060-009-6

Buku ini telah dinilai oleh Badan Standar Nasional Pendidikan (BSNP) dan telah dinyatakan layak sebagai buku teks pelajaran berdasarkan Peraturan Menteri Pendidikan Nasional Nomor 45 Tahun 2008 tanggal 15 Agustus 2008 tentang Penetapan Buku Teks Pelajaran yang Memenuhi Syarat Kelayakan untuk digunakan dalam Proses Pembelajaran.

HET (Harga Eceran Tertinggi) Rp. 13,794.00