

BAB II

LANDASAN TEORI

2.1 Pengantar

Pada bab ini akan dijelaskan mengenai uraian dasar tentang teori yang digunakan dalam pembuatan aplikasi *SMS Server* menggunakan J2SE. Ada dua hal yang akan dijelaskan disini. Pertama tentang SMS (*Short Messaging Service*) dan kedua landasan teori tentang pembuatan aplikasi baik tool yang akan digunakan, yaitu J2SE maupun sistem jaringan yang ada dalam aplikasi ini.

2.2 Short Messaging Service (SMS)

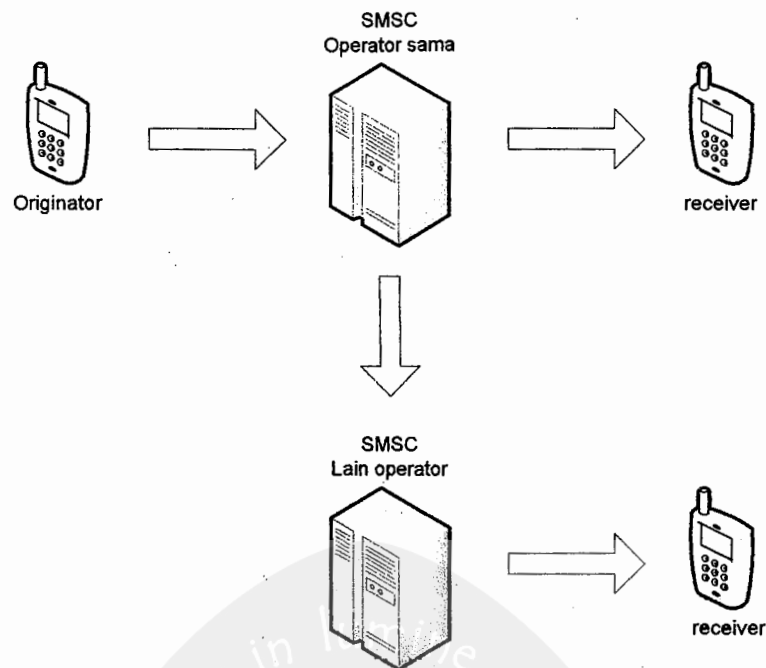
SMS atau *Short Messaging Service* merupakan salah satu media yang paling populer dan yang paling banyak digunakan sekarang ini. Ini dikarenakan biaya yang dikeluarkan cukup murah dan prosesnya cepat, langsung menuju tujuan. SMS merupakan salah satu fitur dari GSM (*Global System for Mobile Communications*), yang dikembangkan dan distandarisasi oleh ETSI (*European Telecommunication Standards Institute*). Pesan-pesan (dari SMS) ini dikirim dengan cara *store and forward*. Untuk SMS *point-to-point*, sebuah pesan dapat dikirim ke pelanggan lainnya melalui layanan dan laporan penerimaannya dikirimkan kepada pengirim. SMS pertama dipercaya telah terkirim pada bulan Desember 1992 dari sebuah PC (*Personal Computer*) ke sebuah ponsel pada jaringan Vodafone GSM di Inggris (Simon Buckingham, 2001). Meskipun telah banyak fitur-fitur yang

ditawarkan dari GSM seperti EMS, MMS, dan GPRS, keberadaan jasa dan industri yang menggunakan SMS semakin lama semakin banyak dijumpai. Hal ini dikarenakan oleh faktor hardware yang semakin hari semakin terjangkau.

Adapun jasa atau servis yang menggunakan SMS diantaranya adalah perbankan, seperti *SMS Banking*. Dalam *SMS Banking* ini kita hanya tinggal mengirimkan SMS untuk mengetahui saldo dari tabungan kita. Kemudian contoh lain adalah *Contact Center*, dimana mereka memanfaatkan SMS untuk meningkatkan servis level kepada pelanggannya. Pada bidang industri, seperti pemanfaatan SMS pada aplikasi GPS, yang dapat memonitor posisi langsung ke ponsel pengguna. Kemudian di pabrik-pabrik yang memanfaatkan SMS sebagai *alerting* program dari mesin pabrik bila sewaktu-waktu mengalami kerusakan. Pada industri IT (*Information Technology*), data center yang memanfaatkan SMS untuk mengawasi kinerja dari server. Serta masih banyak lagi jasa dan industri yang menggunakan media ini.

2.2.1 Cara Kerja SMS

Pada saat kita mengirim pesan SMS dari ponsel (*mobile originated*) pesan tersebut tidak langsung dikirimkan ke ponsel tujuan (*mobile terminated*), akan tetapi dikirimkan terlebih dahulu ke *SMS Center* (SMSC), baru kemudian pesan tersebut diteruskan ke ponsel tujuan.



Gambar 2.1 Skema cara kerja SMS

Dengan adanya SMSC ini kita dapat mengetahui status dari pesan SMS yang telah dikirim, apakah telah sampai atau gagal diterima oleh ponsel tujuan. Apabila ponsel tujuan dalam keadaan aktif dan dapat menerima pesan SMS yang dikirim, ia akan mengirimkan kembali pesan konfirmasi ke SMSC yang menyatakan bahwa pesan telah diterima. Kemudian SMSC mengirimkannya kembali status tersebut kepada si pengirim. Jika ponsel tujuan dalam keadaan mati, pesan yang kita kirimkan akan disimpan pada SMSC sampai *period-validity* terpenuhi.

Untuk dapat mengirim dan menerima pesan SMS, kita harus melakukan koneksi ke SMSC. Ada beberapa cara untuk melakukan koneksi ke SMSC, antara lain dengan cara:

1. Menggunakan terminal baik berupa GSM modem atau Handphone

Cara ini adalah cara yang paling mudah tetapi memiliki kekurangan antara lain jumlah pesan yang dikirim per menit sangat terbatas (sekitar 6-10 pesan per menit). Untuk mengantisipasi hal ini biasanya digunakan lebih dari satu terminal.

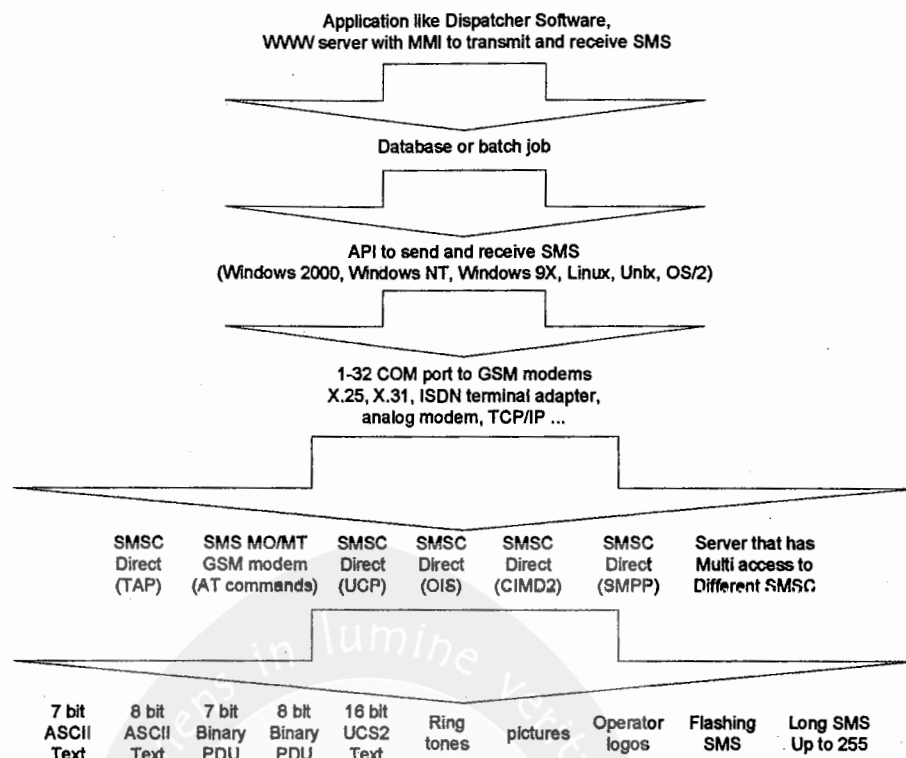
2. Koneksi langsung ke SMSC

Dengan melakukan koneksi langsung ke SMSC kita dapat mengirimkan pesan dalam jumlah banyak, dapat mencapai sekitar 600 sms per menit bergantung pada kapasitas dari SMSC itu sendiri. Untuk melakukan koneksi ke SMSC diperlukan protokol penghubung. Protokol penghubung yang umum digunakan adalah UCP, SMPP, CIMD2, OIS dan TAP. Masing-masing operator GSM menyediakan tipe-tipe protokol yang berbeda-beda.

3. Menggunakan software bantu

Saat ini banyak vendor telekomunikasi menawarkan software bantu untuk melakukan koneksi ke SMC, dari yang bersifat freeware, open source, sampai dengan komersial.

Pemilihan koneksi ke SMSC biasanya disesuaikan dengan jumlah pesan SMS yang akan dikirim.



Gambar 2.2 Tipe koneksi SMS Center
(Sumber: Gunawan, Ferry, 2003)

Setelah kita mengetahui cara kerja SMS, kemudian dilanjutkan dengan *SMS Center*, maka ada lagi yang perlu kita ketahui, yaitu alamat dari *SMS Center*. Berikut adalah daftar SMSC pada beberapa operator GSM di Indonesia:

Tabel 2.1 Daftar SMSC

Operator GSM	Nomor SMSC
Satelindo	62816124
Excelcomindo	62818445009
Telkomsel	6281100000
IM3	62855000000

2.2.2 Mengirim dan Menerima SMS

Dalam mengirim dan menerima SMS sebenarnya ada dua mode untuk mengirim dan menerima SMS, yaitu mode text dan mode PDU (*Protocol Data Unit*). Akan tetapi, sistem mode text tidak didukung oleh semua operator GSM maupun terminal. Pada terminal, anda dapat mengeceknya menggunakan perintah "AT+CMGF=1". Jika hasilnya error, dapat dipastikan bahwa terminal anda tidak mendukung mode teks. Demikian juga sebaliknya.

Mode text atau *Text mode* adalah cara termudah untuk mengirim pesan. Pada mode text pesan yang kita kirim tidak dilakukan konversi. Teks yang dikirim tetap dalam bentuk aslinya dengan panjang mencapai 160 (7 bit default alphabet) atau 140 (8 bit) karakter. Sesungguhnya, mode text adalah hasil encode yang direpresentasikan dalam format PDU. Kelemahan dari mode Text ini adalah kita tidak dapat menyisipkan gambar dan nada dering ke dalam pesan yang akan kita kirimkan serta terbatasnya tipe *encoding*.

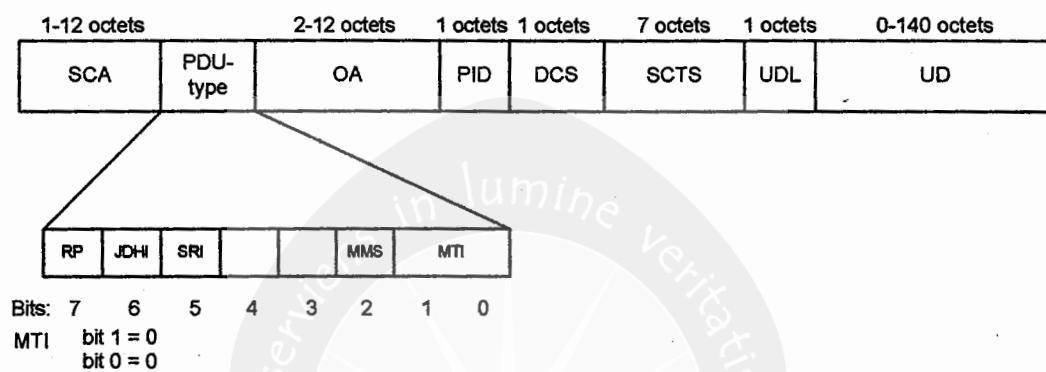
PDU Mode adalah format *message* dalam *heksadesimal octet* dan *semi-decimal octet* dengan panjang mencapai 160 (7 bit default alphabet) atau 140 (8 bit) karakter. Kelebihan menggunakan mode PDU adalah kita dapat melakukan enkoding sendiri yang tentunya kita juga harus didukung dengan hardware dan operator GSM, melakukan kompresi data, menambahkan nada dering dan gambar pada pesan yang akan dikirim. Beberapa tipe enkoding yang umum digunakan adalah "PCCP437", "PCDN", "8859-1", "IRA", dan "GSM". Anda dapat mengeceknya dengan perintah "AT+CSCS". Kita juga dapat menambahkan header kedalam pesan yang akan dikirim, seperti

timestamp, nomor SMSC dan meta-informasi lainnya. (Gunawan, Ferry, 2003)

2.2.3 SMS Deliver PDU

SMS Deliver PDU adalah terminal menerima pesan yang datang / masuk dari SMSC dalam format PDU.

Berikut adalah gambar dari skema format SMS Deliver PDU



Gambar 2.3 Skema format SMS deliver PDU.

(Sumber: Gunawan, Ferry, 2003)

Message Asli:	"terima sms", "terima" + spasi + "sms"
PDU:	059126181642240C912618859504820000305031614034820AF4B23CDD0E83E6ED39

Gambar 2.4 Contoh PDU pada saat menerima sms.

Diatas adalah contoh dari bentuk PDU pada saat terminal menerima SMS yang datang / masuk dari SMSC, dengan penjelasan sebagai berikut:

1. Service Center Address (SCA)

SCA memiliki tiga komponen utama, yaitu len, type of number dan BCD Digits.

Tabel 2.2 Tabel Keterangan SCA untuk *SMS Deliver* PDU

Octet	Keterangan	Nilai
Len	Panjang informasi SMSC dalam octet	05
Type of number	Type of address dari SMSC 81h = lokal format 91h = internasional format	91
BCD Digits	Nomor SMSC. Jika panjangnya ganjil, pada akhir karakter tambahkan 0F hexa	26181642

(Sumber: Gunawan, Ferry, 2003)

Berikut adalah contoh penulisan nomor SMSC untuk masing-masing operator di Indonesia:

Tabel 2.3 Tabel Nomor SMSC dalam format PDU

Operator	Nomor SMSC	Format dalam PDU
Satelindo	6281624	05 91 26 18 16 42
Excelcomindo	62818445009	07 91 26 18 48 54 00 F9
Telkomsel	6281100000	06 91 26 18 01 00 00
IM3	62855000000	05 91 26 58 05 00 00 F0

(Sumber: Gunawan, Ferry, 2003)

2. Protocol Data Unit (PDU) Type

Nilai default dari PDU untuk *SMS-Deliver* adalah 04h. Pada contoh diatas, PDU type adalah 24 yang memiliki arti sebagai berikut:

Bt.no	7	6	5	4	3	2	1	0
Nama	RP	UDHI	SRI	<none>	<none>	MMS	MTI	MTI
Nilai	0	0	1	0	0	1	0	0

3. Originator Address (OA)

Berikut adalah tabel keterangan mengenai *Originator Address*.

Tabel 2.4 Tabel Keterangan OA

Octet	Keterangan	Nilai
Len	Panjang nomor Originator Address	0C
Type of number	Tipe dari Originator Address 81h = lokal format 91h = internasional format	91
BCD Digits	Nomor Originator Address. Jika panjangnya ganjil, pada akhir karakter tambahkan 0F hexa	261885980482

(Sumber: Gunawan, Ferry, 2003)

Berikut adalah contoh format penulisan OA:

Nomor OA	Format dalam PDU
ABCDEFGHJKLMNOPQRST	14 81 BA DC FE HG JI LK NM PO RQ ST
+ABCDEFGHJKLM	0C 91 BA DC FE HG JI LK

4. Protocol Identifier (PID)

Nilai default dari PID adalah 0, dimana nilai 0 ini menunjukkan "Standard-Text-SMS". Pada contoh diatas adalah 00.

5. Data Coding Scheme (DCS)

Pada contoh diatas DCS-nya adalah 00. DCS juga menentukan *message class* seperti pada tabel 2.5 dibawah ini.

Tabel 2.5 Tabel penentuan Message Class DCS

Nilai (Hexa)	Character Coding	Message Class
00	default (7 bit)	No class
F0	default (7 bit)	Class 0 (immediate display)
F1	default (7 bit)	Class 1 (Mobile Equipment-specific)
F2	default (7 bit)	Class 2 (SIM specific message)
F3	default (7 bit)	Class 3 (Terminate Equipment-specific)
F4	8-bit	Class 0 (immediate display)
F5	8-bit	Class 1 (Mobile Equipment-specific)
F6	8-bit	Class 2 (SIM specific message)
F7	8-bit	Class 3 (Terminate Equipment-specific)

(Sumber: Gunawan, Ferry, 2003)

Hal yang perlu diperhatikan disini adalah pada beberapa ponsel, *message* dengan *class* 0 dengan enkoding 7 bit berupa flash SMS. Sedangkan dengan enkoding 16 bit *Unicode* (ucs2), *message* yang didahului "0001" dengan *class* 0 berupa *blinking flash SMS*.

6. Service Center Time Stamp (SCTS)

Pada contoh penulisan PDU Deliver diatas, SCTS adalah 30503161403482, yang berarti:

Tahun	30 menjadi 2003
Bulan	50 menjadi 05, Mei
Tanggal	31 menjadi 13
Jam	16 menjadi 16
Menit	40 menjadi 04
Detik	34 menjadi 43
Detik	82 menjadi 28, dimana 1 unit = 15 menit (15 x 28)/60 = 7, menjadi GMT +07:00

Penulisan diatas dibalik (*swapped nibble presentation*) 30 menjadi 03 (menunjukkan tahun 2003), 50 menjadi 05 (menunjukkan bulan 05), 31 menjadi 13 (menunjukkan tanggal 13), 61 menjadi 16 (menunjukkan jam 16), 40 menjadi 04 (menunjukkan menit 04), 34 menjadi 43 (menunjukkan detik 43), 82 menjadi 28 (menunjukkan zona waktu dimana 1 unit adalah 15 menit, jadi $(15 \times 28)/60 = 7$ menjadi GMT +07:00). Jadi arti SCTS diatas sama dengan 13 Mei 2003 pukul 16:04:43 GMT+07:00.

7. User Data Length (UDL)

Pada contoh diatas, UDL adalah 0A. UDL menunjukkan panjang User Data (UD).

8. User Data (UD)

Pesan "terima sms" yang dikodekan menjadi 7 bit default alphabet (septet) harus di-encode menjadi 8 bit (octet) untuk *SMS-Transfer*. Pada contoh diatas, UD adalah "F4B23CDD0E83E6ED39". Tabel 2.6 dibawah ini menunjukkan car pengkodean dengan menggunakan batuan Tabel 2.7.

Tabel 2.6 Pengkodean 8 bit menjadi 7 bit

Nilai (Hexa)	desimal	Octet (8 bit)	Septet (7 bit)	Karakter
F4	244	1 1110100	1110100	t
B2	178	10 110010	110010 1	e
3C	60	001 11100	11100 10	r
DD	221	1101 1101	1101 001	i
0E	14	00001 110	110 1101	m
83	131	100000 11	11 00001	a
E6	230	1110011 0	0 100000	<spasi>
ED	237	1 1101101	1110011	s
39	57	00 111001	1101101	m
			111001 1	s

(Sumber: Gunawan, Ferry, 2003)

Tabel 2.7 Default Alphabet (7 bit)

					b7	0	0	0	0	1	1	1	1
					b6	0	0	1	1	0	0	1	1
					b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1			0	1	2	3	4	5	6	7
0	0	0	0	0	@	Δ	SP	0	i	P	ı	p	
0	0	0	1	1	£	_	!	1	A	Q	a	q	
0	0	1	0	2	\$	Φ	“	2	B	R	b	r	
0	0	1	1	3	¥	Γ	#	3	C	S	c	s	
0	1	0	0	4	è	Λ	□	4	D	T	d	t	
0	1	0	1	5	é	Ω	%	5	E	U	e	u	
0	1	1	0	6	ù	Π	&	6	F	V	f	v	
0	1	1	1	7	ì	Ψ	‘	7	G	W	g	w	
1	0	0	0	8	ø	Σ	(	8	H	X	h	x	
1	0	0	1	9	Ç	Θ	)	9	I	Y	i	y	
1	0	1	0	10	LF	Ξ	*	:	J	Z	j	z	
1	0	1	1	11	Ø	ı)	+	;	K	Ä	k	ä	
1	1	0	0	12	ø	Æ	,	<	L	Ö	l	ö	
1	1	0	1	13	CR	æ	-	=	M	Ñ	m	ñ	
1	1	1	0	14	Å	ß	.	>	N	Ü	n	ü	
1	1	1	1	15	å	Ê	/	?	O	§	b	ã	

(Sumber: Gunawan, Ferry, 2003)

Pada tabel diatas b1...b7 menunjukkan posisi atau urutan-bit pada septet seperti yang digambarkan pada tabel 2.8 dibawah ini.

Tabel 2.8 posisi b1 s/d b7 pada septet

b7	b6	b5	b4	b3	b2	b1
xa	xb	xc	xd	xe	xf	xg

(Sumber: Gunawan, Ferry, 2003)

Kemudian susunan bit dari satu karakter pada satu octet adalah sebagai berikut:

Tabel 2.9 Susunan bit 1 octet

Octet	bit							
	7	6	5	4	3	2	1	0
1	0	1a	1b	1c	1d	1e	1f	1g

(Sumber: Gunawan, Ferry, 2003)

Sedangkan susunan bit dari dua karakter pada dua octet adalah sebagai berikut:

Tabel 2.10 Susunan bit 2 octet

Octet	bit							
	7	6	5	4	3	2	1	0
1	2g	1a	1b	1c	1d	1e	1f	1g
2	0	0	2a	2b	2c	2d	2e	2f

(Sumber: Gunawan, Ferry, 2003)

Proses diatas terus berlangsung sehingga pada 140 octet dapat menampung $(140 \times 8) / 7 = 160$ karakter.

2.2.4 SMS Submit PDU

SMS Submit PDU adalah pesan dikirim dari terminal ke SMSC dalam format PDU.

Oleh karena panjang dari SMSC adalah 0, *field-field* yang lain dapat diabaikan. SMSC yang digunakan adalah SMSC yang terdapat pada SIM Card berdasarkan perintah "AT+CSCA".

2. Protocol Data Unit (PDU) type

Nilai default dari PDU untuk *SMS-Submit* adalah 11h. Pada contoh diatas, PDU Type adalah 11 yang memiliki arti:

Bit no.	7	6	5	4	3	2	1	0
Nama	RP	UDHI	SRR	VPF	VPF	RD	MTI	MTI
Nilai	0	0	0	1	0	0	0	1

3. Message Reference (MR)

Pada contoh diatas, MR adalah 00.

4. Destination Address (DA)

Penulisan format Destination Address sama dengan Originator Address.

Tabel 2.12 Tabel Keterangan DA

Octet	Keterangan	Nilai
Len	Panjang nomor Destination Address	0C
Type of number	Tipe dari Destination Address 81h = lokal format 91h = internasional format	91
BCD Digits	Nomor Destination Address. Jika panjangnya ganjil, pada akhir karakter tambahkan 0F hexa	261885980482

(Sumber: Gunawan, Ferry, 2003)

5. Protocol Identifier (PID)

Penjelasan mengenai *Protocol Identifier* pada *SMS Submit* PDU ini sama dengan *Protocol Identifier*

pada *SMS Deliver* PDU diatas. Pada contoh disini PID-nya adalah 00.

6. Data Coding Scheme (DCS)

Penjelasan mengenai *Data Coding Scheme* pada *SMS Submit* PDU ini sama dengan *Data Coding Scheme* pada *SMS Deliver* PDU. Pada Contoh disini DCS-nya adalah 00.

7. Validity Period (VP)

Pada contoh diatas, VP-nya adalah AA, atau 170d, $170-166 = 4$ hari.

Nilai VP	Nilai validitas periode
0 - 143	$(VP + 1) * 5$ menit(interval 5 menit hingga 12 jam)
144 - 167	12 jam + $((TP - VP - 143) * 30)$ menit
168 - 196	$(VP - 166) * 1$ hari
197 - 255	$(VP - 192) * 1$ minggu

8. User Data Length (UDL)

Penjelasan mengenai UDL untuk *SMS Submit* PDU ini sama dengan UDL untuk *SMS Deliver* PDU. UDL pada contoh disini adalah 09.

9. User Data (UD)

Pesan "kirim sms" dikodekan 7 bit default alphabet (septet) sehingga harus dikodekan menjadi 8 bit (octet), yaitu "EBB43CDD06CDDDB73". Pada setiap octet jika jumlah bit kurang dari 8, maka diambil bit paling kanan pada septet selanjutnya dan ditambahkan pada bagian kiri septet sebelumnya mengikuti kaidah pada tabel 2.9 dan 2.10 diatas. Tabel 2.13 dibawah ini menunjukkan cara pengkodean.

Tabel 2.13 Pengkodean 7 bit menjadi 8 bit

Karakter	desimal	Septet (7 bit)	Octet (8 bit)	Nilai (Hexa)
K	107	1101011	1 1101011	EB
I	105	110100 1	10 110100	B4
R	114	11100 10	001 11100	3C
I	105	1101 001	1101 1101	DD
M	109	110 1101	00000 110	06
<spasi>	32	01 00000	110011 01	CD
S	115	1 110011	1101101 1	DB
M	109	1101101	00 1110011	73
S	115	1110011		

(Sumber: Gunawan, Ferry, 2003)

2.3 SMS Gateway

Setelah kita mengetahui pengertian SMS, cara kerja, dan seluk beluk lainnya mengenai SMS, sekarang kita beranjak ke *SMS Gateway*. Arti dari *Gateway* itu sendiri adalah pintu gerbang, atau mungkin dalam kasus ini adalah jembatan antara PC (karena dalam kasus ini kita menggunakan *SMS Server*, dimana server yang digunakan adalah sebuah PC) dengan operator media komunikasi / operator GSM (karena SMS berada dalam teknologi media komunikasi GSM).

Karena infrastruktur komunikasi seluler berada dalam jaringan mereka sendiri, kemudian *SMS Server* ini juga berada dalam infrastruktur yang berbeda dengan infrastruktur seluler diatas. Oleh karena itu supaya aplikasi *SMS Server* tersebut dapat berjalan, maka diperlukan perantara antara infrastruktur komunikasi seluler dengan infrastruktur *SMS Server* yang nantinya dapat digunakan oleh perseorangan atau perusahaan ini. Hal ini diharapkan supaya aplikasi *SMS Server* ini dapat melakukan aktifitasnya seperti mengirim SMS, menerima SMS, dan memproses SMS.

2.4 Teknologi dan Tool yang dipakai

2.4.1 Konsep Sistem Komputer dan Jaringan

2.4.1.1 Protokol TCP / IP (*Transmission Control Protocol / Internet Protocol*)

Dalam jaringan komputer dikenal adanya istilah protokol. Protokol adalah suatu aturan atau mekanisme di mana dua atau lebih komputer dapat saling berinterkoneksi. Sebuah protokol tentu saja mendefinisikan suatu format paket data yang akan dipertukarkan untuk menunjang mekanisme tersebut.

Salah satu protokol yang paling banyak digunakan saat ini adalah protokol TCP/IP. Protokol ini juga digunakan dalam jaringan terbesar, yaitu internet.

Protokol-protokol TCP/IP dikembangkan pada awal 1980-an dan menjadi protokol-protokol standar untuk ARPAnet pada tahun 1983. TCP/IP adalah serangkaian protokol di mana setiap protokol melakukan sebagian dari keseluruhan tugas komunikasi jaringan. Berikut arsitektur protokol TCP/IP seperti pada gambar 2.1.

Arsitektur Protokol TCP/IP :

<i>Application / Process</i>
<i>Transport / Host to Host</i>
<i>Internetwork</i>
<i>Network Access</i>

Gambar 2.7 : Arsitektur model TCP/IP

Lapisan *Network Access* bertanggung jawab dalam pertukaran data antara *host* dan jaringan, serta untuk pengiriman data antara dua peralatan pada jaringan yang sama. Beberapa standar protokol yang termasuk dalam lapisan ini antara lain Ethernet II, IEEE 802.x, X.25, Frame Relay, dan lain-lain.

Lapisan *Internetwork* disini yaitu Internet Protokol. Internet Protokol merupakan protokol antar jaringan yang diperlukan untuk pengalamatan, fragmentasi diagram antar jaringan dan pengiriman datagram antar jaringan.

Lapisan *Host to Host* ini berfungsi sebagai antarmuka antara proses-proses dan aplikasi lapisan atas ke jaringan dan mengirim pesan lapisan atas antar *host*. Ada dua protokol *host* ke *host* untuk memenuhi kebutuhan proses lapisan atas, yaitu *Transmission Control Protocol* (TCP) dan *User Datagram Protocol* (UDP) (Wahju, RJB Agung dkk, 2002).

Berikut perbandingan TCP dan UDP pada tabel 2.14:

Tabel 2.14 : Perbandingan TCP dan UDP

TCP	UDP
- <i>connection oriented</i>	- <i>connection less</i>
- pengurutan paket yang datang	- tanpa pengurutan paket yang datang
- retransmisi paket yang hilang	- tanpa retransmisi paket yang hilang
- tak ada duplikasi paket	- mungkin terjadi duplikasi paket
- melakukan <i>flow control</i>	- tak melakukan <i>flow control</i>
- mendukung komunikasi <i>multiserver</i> dan <i>multiclient</i> dengan nomor port yang sama (<i>multiplexer</i> dan <i>demultiplexer</i>)	- tidak mendukung
- tak memperhatikan batas rekaman	- memperhatikan batas rekaman
- tak mendukung <i>broadcast</i> dan <i>multicast</i>	- mendukung <i>broadcast</i> dan <i>multicast</i>
- <i>overhead</i> tinggi (header 24 byte)	- <i>overhead</i> relatif rendah (header 8 byte)
- digunakan untuk transmisi yang	- digunakan untuk transmisi yang

membutuhkan kehandalan	berorientasi waktu atau pengiriman pesan secara periodik
---------------------------	--

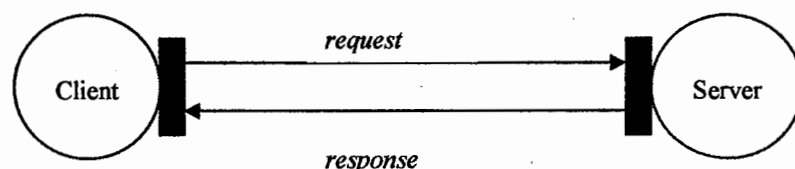
(Sumber :Diktat Praktikum Sistem Operasi dan Jaringan, Universitas Atma Jaya, Yogyakarta)

Lapisan *Application / Process* adalah tempat dilaksanakannya proses kerja yang sesungguhnya. Disini dapat ditemukan program yang menyediakan pelayanan jaringan, seperti *FTP, mail server, telnet* dan sebagainya (Wahju, RJB Agung dkk, 2002).

2.4.1.2 Pemrograman Client / Server

Konsep pemrograman *client/server* cukup sederhana, yaitu sebuah proses/aplikasi *client* melakukan permintaan untuk suatu informasi atau mengirim sebuah perintah ke suatu aplikasi *server*. Aplikasi *server* akan menerima permintaan dari *client*, kemudian memproses berdasarkan permintaan tersebut, dan akan mengembalikan sesuatu ke *client* sebagai suatu hasil dari pemrosesan yang sudah dilakukan.

Sehingga tugas dari *server* adalah melakukan *listen* untuk suatu koneksi, sedangkan *client* mencoba membuat koneksi ke *server*. Setelah koneksi terbentuk, hubungan pertukaran data antara *client* dan *server* diwakili dengan suatu *stream* input dan output, seperti pada gambar 2.8.



Gambar 2.8 : Model *Client/Server*

Ketika dalam pertukaran data antara *client* dan *server* dengan menggunakan protokol TCP/IP, dalam melakukan pengiriman data dari *client* ke *server* atau sebaliknya, digunakanlah suatu bentuk pengalamatan pengiriman data, yaitu **AlamatIP:Port**.

Terdapat dua kelompok dalam pemrograman *client/server* berbasis TCP/IP, yaitu aplikasi *client/server* yang sifatnya implementasi dari suatu protokol yang sudah didefinisikan oleh suatu RFC (*Request For Comments*). Contohnya adalah HTTP, FTP, SMTP, SNMP, IRC, POP3, IMAP dan sebagainya. Kelompok kedua adalah aplikasi *client/server proprietary*. Pada kelompok ini, aplikasi *client/server* tidak perlu mengikuti aturan protokol yang sudah didefinisikan RFC tertentu. Dapat dibuat suatu mekanisme pertukaran data sendiri antara *client* dan *server*.

Untuk menyembunyikan kompleksitas penggunaan protokol pendukung aplikasi *client/server* berbasis TCP/IP, dibuatlah suatu antarmuka dan pustaka kelas atau fungsi yang memberikan kemudahan bagi pemrograman untuk memperkaya aplikasi *client/server* yang dapat berjalan pada jaringan TCP/IP. Antarmuka dan pustaka ini disebut dengan **socket**. (Susanto, Budi, 2003)

2.4.1.3 Socket

Komunikasi antar proses yang terjadi pada mesin yang sama dapat dilakukan dengan menggunakan fasilitas IPC (*Inter Process Communication*), antara lain menggunakan *pipe*, *named-pipe*, *shared memory*, *message queue*, dsb. Fasilitas IPC tersebut diatas tidak dapat mendukung komunikasi antara dua proses yang terletak

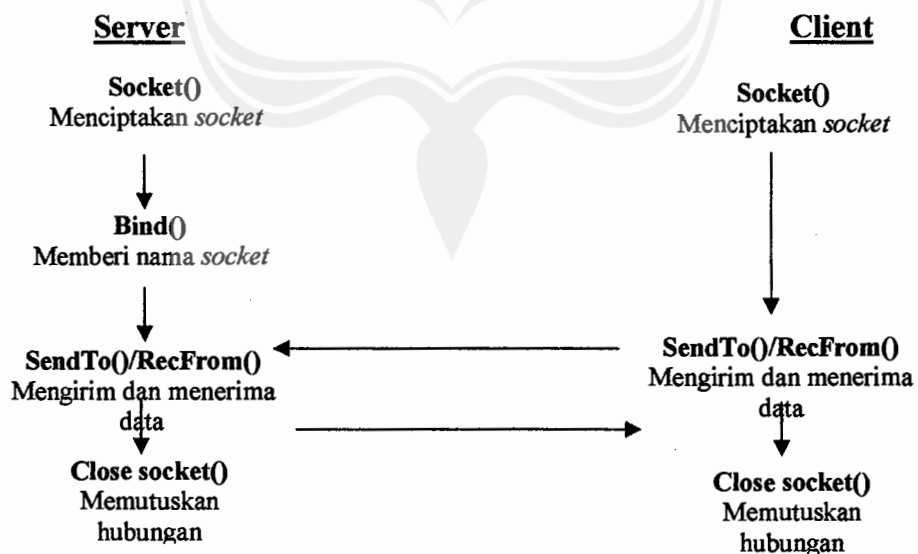
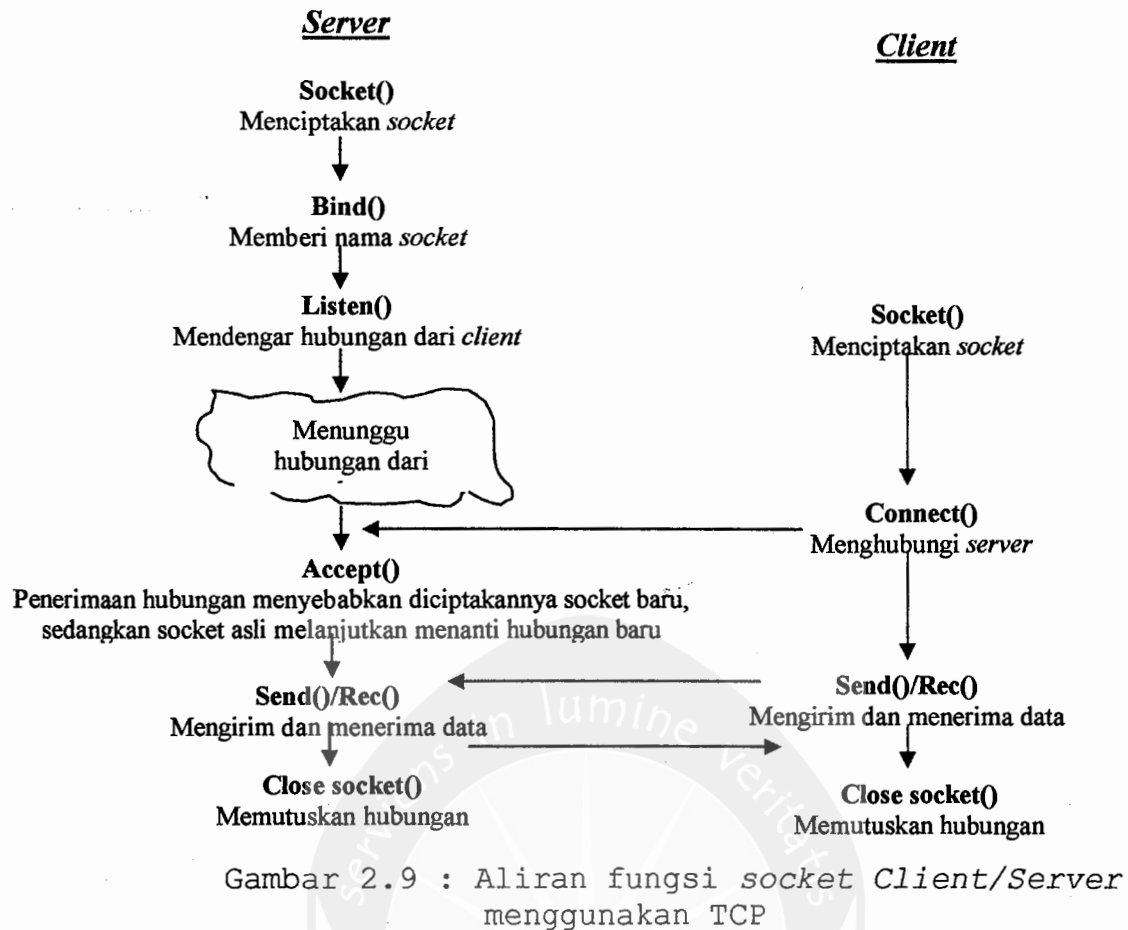
pada mesin yang berbeda. Komunikasi di antara dua proses yang terletak pada mesin yang berbeda dapat dilakukan melalui jaringan yang menghubungkan kedua mesin. *Socket* merupakan fasilitas IPC dalam jaringan. Berdasarkan protokolnya, *socket* dapat dibedakan menjadi:

1. *SOCK_STREAM*, menyediakan komunikasi sistem *byte-to-byte*, dapat digunakan dua arah (TCP).
2. *SOCK_DGRAM*, digunakan untuk transmisi datagram, juga dapat digunakan dua arah (UDP).
3. *SOCK_RAW*, disediakan untuk user yang menginginkan *high level* kontrol atas transmisi pesan.

Berdasarkan keaktifannya, *socket* dapat dibedakan menjadi :

1. *Socket* aktif, digunakan pada sisi *client*, karena proses komunikasi akan diawali oleh sisi *client* dengan menghubungi sisi *server*.
2. *Socket* pasif, digunakan pada sisi *server*, karena *server* akan menunggu adanya permintaan sambungan dari luar (*client*).

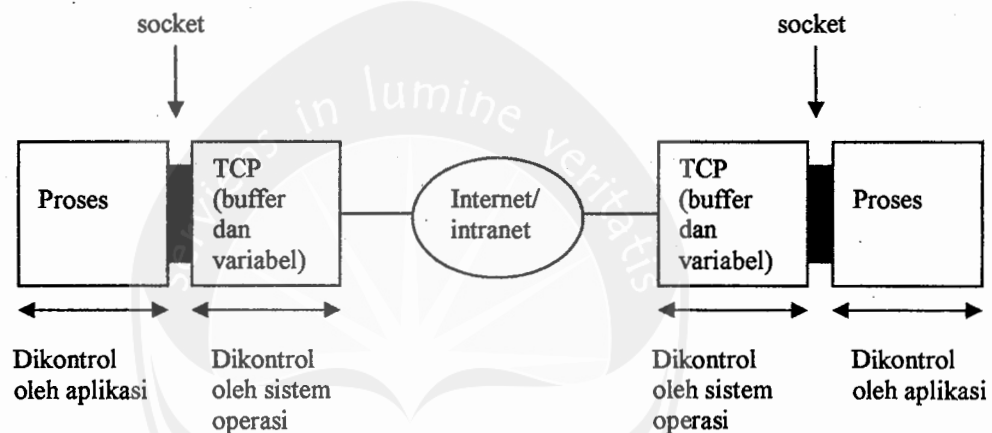
Untuk dapat menggunakan *socket* dalam komunikasi proses antar mesin dalam jaringan diperlukan cara pemanggilan (*system call*) fungsi untuk *socket* (Stevens, 1991). Berikut gambar 2.9 dan 2.10 menggambarkan aliran fungsi *socket client/server* dengan menggunakan TCP dan UDP :



(Sumber: Wahyu, RJB Agung dkk, 2003)

2.4.1.4 Pemrograman Socket Connection Oriented

Seperti yang telah diuraikan sebelumnya, untuk membangun aplikasi jaringan berbasis TCP/IP dengan tipe koneksi *Connection Oriented*, akan digunakan protokol transport TCP. Ketika sebuah proses server ataupun client saling berinterkoneksi (bertukar data), masing-masing akan terhubung atau menggunakan protokol TCP melalui sebuah *socket*. Sebuah proses ketika berhubungan dengan TCP akan memiliki nomor Port di samping alamat komputernya.



Gambar 2.11 : Komunikasi antar proses dengan menggunakan Socket TCP

Pada gambar 2.11 menunjukkan suatu skema hubungan antara proses aplikasi dengan TCP lokal dan juga hubungan antara dua komputer dengan protokol TCP. *Socket* berperan menghubungkan antara suatu aplikasi dengan protokol TCP lokal. Dengan kata lain, sebuah aplikasi jaringan ketika akan mengirimkan data ke komputer atau proses lain menggunakan protokol TCP, harus menggunakan *socket*, agar data yang dikirimkan dapat diterima oleh TCP. Berikutnya adalah tugas TCP yang dikontrol oleh sistem operasi untuk dapat

mengantarkan data yang dikirimkan ke TCP di komputer atau proses lainnya. (Susanto, Budi, 2003)

Koneksi Socket TCP pada Java

Ada 4 langkah dalam memprogram server atau *client* untuk menggunakan *socket*:

1. Membuka socket

Pada *client*, hal ini dilakukan sebagai berikut:

```
Socket myKlien = null;
try {
    myKlien = new Socket("host", NomorPort);
} catch (UnknownHostException uhe) {
    uhe.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Pada *server*:

```
public static final int NomorPort = 2000;
ServerSocket Layanan = null;
try {
    Layanan = new ServerSocket(NomorPort);
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

Selain itu, pada *server* juga harus dibuat sebuah objek *socket* dari kelas *ServerSocket* untuk mendengar dan menerima koneksi dari klien, sebagai berikut:

```
Socket layananSocket = null;
try {
    layananSocket = Layanan.accept();
} catch (IOException iex) {
    iex.printStackTrace();
}
```

2. Membuat data input stream

Untuk membuat *input stream* pada *client*, dapat digunakan kelas *BufferedReader* untuk menerima respon dari server.

```
BufferedReader is = null;
try {
    is = new BufferedReader(new InputStreamReader
        (myKlien.getInputStream()));
}
```

```

    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}

```

Pada server juga dapat digunakan `BufferedReader` untuk menerima input dari *client*.

```

BufferedReader is = null;
try {
    is = new BufferedReader(new InputStreamReader
(Layanan.getInputStream()));
} catch (IOException ioe) {
    ioe.printStackTrace();
}

```

3. Membuat data output stream

Pada *client*, dapat digunakan kelas `DataOutputStream` untuk mengirimkan data ke *socket server*.

```

DataOutputStream os = null;
try {
    os = new DataOutputStream(myKlien.getOutputStream());
} catch (IOException ix) {
    ix.printStackTrace();
}

```

Pada server:

```

DataOutputStream os = null;
try {
    os = new DataOutputStream(Layanan.getOutputStream());
} catch (IOException ie) {
    ie.printStackTrace();
}

```

4. Menutup *socket*

Pada *client*:

```

try {
    os.close();
    is.close();
    myKlien.close();
} catch (IOException io) {
    io.printStackTrace();
}

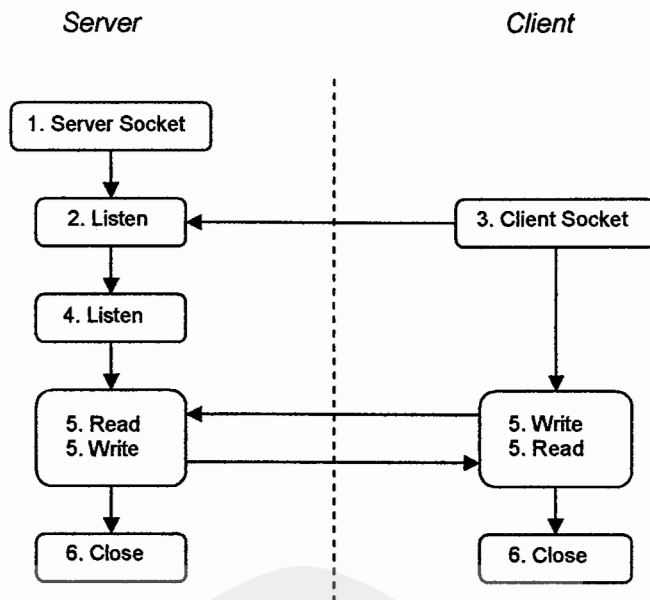
```

Pada server:

```

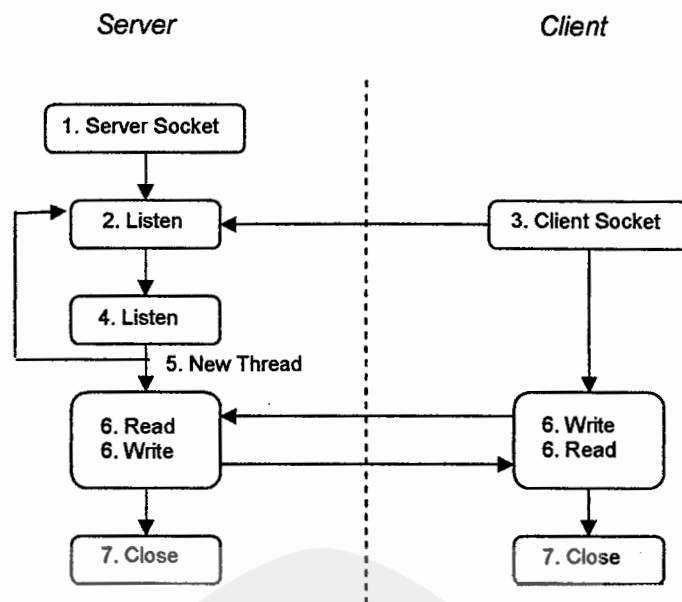
try {
    os.close();
    is.close();
    layananSocket.close();
} catch (IOException ic) {
    ic.printStackTrace();
}

```



Gambar 2.12 : Aplikasi *Client-Server* dengan *single-thread server*

Seperti pada gambar 2.12, aplikasi *server* yang dibuat merupakan *single-thread server* yaitu aplikasi *server* yang hanya bisa menangani satu *client* pada satu waktu. Agar *server* memungkinkan untuk menangani lebih dari satu *client* maka perlu adanya pemrograman *multithreading*. Istilah *multithreading* secara prinsip tersusun dari dua kata, yaitu *multi* dan *thread*. *Thread* lebih ditujukan untuk sebuah aliran kontrol dalam sebuah program. Dalam suatu aplikasi dapat berjalan *thread* lebih dari satu, dan dari semua *thread* menggunakan *space address* memori yang sama, sehingga satu *thread* dengan *thread* lain dalam satu program dapat saling berbagi variabel publik. Aplikasi *client/server* dengan *multithread server* dapat dilihat pada gambar 2.13.



Gambar 2.13 : Aplikasi *Client-Server* dengan *multi-thread server*

(Sumber : Wahju, RJB Agung dkk, 2003)

2.4.2 Java

Java adalah sebuah bahasa pemrograman berorientasi objek, yang dikembangkan oleh *Sun Microsystems*. Dikembangkan pada tahun 1995 dengan versi pertama java 1.0. versi pertama tersebut diluncurkan ke masyarakat umum pada tahun 1996 bersama dengan Netscape Navigator 2.0. Dari sejak pertama diciptakan, Java memang dirancang untuk diterapkan dalam lingkungan jaringan komputer (internet), dengan segala kemampuan yang dimilikinya.

Sebagai Bahasa pemrograman, Java dikenal sebagai bahasa pemrograman tingkat tinggi dengan fitur-fitur utama antara lain (Wicaksono, 2002):

1. Sederhana

Pembuatan program dengan Java dapat dipelajari secara cepat jika telah memahami konsep dasar pemrograman berorientasi objek.

2. Berorientasi Obyek

Java adalah murni berorientasi objek. Java bukan turunan langsung dari bahasa pemrograman manapun, dan juga sama sekali tidak kompatibel dengan bahasa pemrograman lainnya. Pada Java semuanya adalah objek secara murni, dengan model yang sederhana dan mudah dikembangkan. Tidak seperti sistem berorientasi objek yang lain, Java memiliki hierarki objek yang sederhana dan mudah diatur, dengan model antarmuka yang dinamis. Dengan memahami gaya dan konsep pemrograman berorientasi objek, akan sangat membantu dalam mempelajari bagaimana membuat program dengan Java.

3. Aman dan Handal

Java dirancang sebagai bahasa pemrograman yang handal dan aman. Aplikasi-aplikasi yang dibangun dengan bahasa Java sangat handal dengan manajemen memori yang baik. Aplikasi Java juga dikenal sangat *secure*. Kasus-kasus seperti *buffer overflow* yang umumnya menjadi lubang keamanan aplikasi-aplikasi berbasis C/C++ tidak terjadi di Java, karena pengaturan keamanannya yang lebih baik. Java memang dirancang dalam sistem yang baru, ketika issue keamanan internet telah menjadi sorotan. Ini membuat Java terhindar dari masalah-masalah yang timbul pada sistem lama, yang ketika dibuat tidak pernah

dirancang dengan konsep keamanan internet. Java tidak pernah memiliki fasilitas dan kemampuan yang tidak aman, sampai perlu ditangani secara khusus untuk pengamanannya.

4. Kokoh

Java membatasi beberapa hal pokok, agar dapat menemukan kesalahan lebih cepat saat mengembangkan program. Banyak kesalahan yang susah dilacak dan situasi jalannya program yang tidak dapat diulangi, dapat dibuat sederhana dengan Java. Kemampuan untuk memastikan apa yang telah ditulis akan berjalan semestinya dalam kondisi apapun adalah salah satu kemampuan penting Java. Java datang dengan kemampuan pengaturan memori yang telah dikembangkan dan penanganan eksepsi secara terpadu.

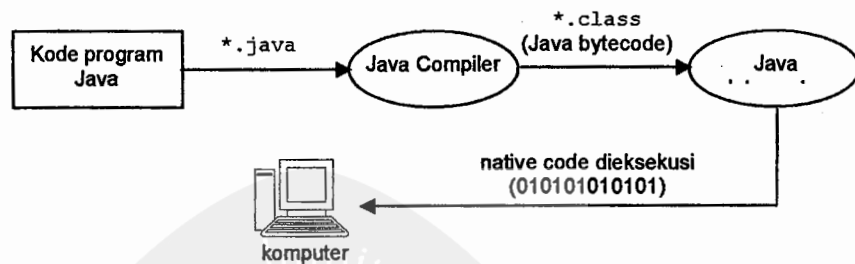
5. Interaktif

Java dirancang memenuhi kebutuhan untuk menciptakan program jaringan yang interaktif. Java memiliki beberapa kemampuan yang memungkinkan program melakukan beberapa hal pada saat bersamaan, tanpa harus kesulitan menangani proses yang akan terjadi selanjutnya. Hal ini memungkinkan pembuatan sistem yang interaktif yang berjalan secara halus.

6. Netral terhadap berbagai Arsitektur (*portable*)

Program Java hanya perlu ditulis sekali, dan dapat dijalankan di semua *platform* tanpa mempedulikan arsitekturnya, apakah Intel®, AMD®, SunSparc®, atau PowerPC®, tanpa perlu dikompilasi ulang. *Java Compiler* yang digunakan untuk

mengkompilasi kode program Java menghasilkan kode yang netral terhadap semua arsitektur perangkat keras, yang disebut dengan *Java bytecode*. Aplikasi-aplikasi hasil kompilasi Java pada dasarnya berupa *Java bytecode* dan bukan *native code* sebagaimana file-file *.exe pada Windows®/DOS (Lihat gambar 2.5).



Gambar 2.14 Proses kompilasi dan eksekusi pada program Java

Aplikasi Java (*.class) merupakan *Java bytecode* yang berjalan di atas *Java Virtual Machine (JVM)*. *Java bytecode* bisa dianggap sebagai kode-kode mesin dari JVM, yang kemudian JVM lah yang akan menginterpretasikan kode-kode tersebut ke *native code* atau kode mesin dari arsitektur yang bersangkutan.

7. Terinterpretasi dan berkinerja tinggi

Java dirancang untuk tetap berkinerja baik pada CPU yang tidak terlalu kuat.

8. Mudah dipelajari

Meskipun lebih rumit daripada bahasa scripting, Java lebih sederhana untuk dipelajari dan ditulis daripada lingkungan pemrograman lain. Karena model objek bersifat sederhana,

maka akan cepat terbiasa dengan gaya pemrograman berorientasi objek (Anonim, 2003).

Sebagai sebuah *platform*, Java terdiri atas dua bagian utama (Wicaksono, 2002), yaitu :

1. *Java Virtual Machine*
2. *Java Application Programming Interface* (Java API)

Pada produksi pertama Java, JDK (*Java Development Kit*) yang digunakan adalah JDK versi 1.0.2. JDK merupakan sekumpulan program dan *library* Java yang digunakan untuk menjalankan dan mengembangkan program Java. Pada rilis selanjutnya, yakni pada versi 1.1, JDK dipecah menjadi dua bagian, yakni JRE (*Java Runtime Environment*) yang dikhususkan untuk menjalankan program-program Java dan JSDK (*Java Perangkat lunak Development Kit*) yang terdiri atas paket-paket yang bisa digunakan untuk mengkompilasi program-program dengan bahasa Java, sekaligus menjalankannya.

Kemudian pada perkembangannya Sun Microsystems memperkenalkan versi terbaru dari bahasa Java yaitu versi 1.2 atau yang lebih dikenal Java 2. Java 2 terdiri atas 3 katagori yaitu (Wicaksono, 2002):

1. *Java 2 Standard Edition* (J2SE)

Kategori ini digunakan untuk mengembangkan aplikasi-aplikasi pada personal computer.

2. *Java 2 Enterprise Edition* (J2EE)

Kategori ini digunakan untuk mengembangkan aplikasi-aplikasi Java pada lingkungan *enterprise*, dengan menambah fungsionalitas Java semacam EJB (*Enterprise Java Bean*), Java Corba,

Servlet, JSP dan Java XML (*Extensible Markup Language*).

3. *Java 2 Micro Edition (J2ME)*

Kategori ini digunakan untuk mengembangkan aplikasi - aplikasi Java pada *mobile device*.

Namun saat ini selain 3 katagori di atas terdapat juga *Java Card*, kategori khusus yang digunakan untuk mengembangkan aplikasi pada *smart cart* seperti aplikasi kartu telepon *chip*, Card pada ponsel. *mobile banking* dan pemrograman *SIM Card* pada ponsel.

