2. TEORI PENUNJANG

2.1. *Short Message Service (SMS)*

Menurut *Short Message Service (SMS)* merupakan salah satu layanan penting dalam jaringan *Global System for Mobile Communication* (GSM). SMS adalah sebuah teknologi yang memungkinkan untuk menerima atau mengirimkan pesan antar telepon bergerak. Teknologi ini pada awalnya menjadi suatu standar untuk telepon *wireless* yang berbasis GSM. Namun teknologi lain seperti CDMA pun memasukkan SMS ini sebagai fitur standar mereka.

Fungsi SMS sebagai alat pengirim pesan singkat sebenarnya bukan merupakan solusi dari hasil pemikiran yang serius. Namun demikian pada akhirnya SMS menjadi sukses secara tak terduga sebagai layanan *messaging* paling populer di dunia. Hal ini tentunya memberikan pendapatan ekstra bagi operator selular yang akan memperoleh bayaran untuk tiap kiriman SMS melalui jaringannya. Sebuah sukses yang tidak disengaja, yang bahkan melebihi fungsi asli sebuah *mobile phone*, sebagai perangkat komunikasi bergerak berbasis suara. Keberhasilan dan popularitas SMS antara lain disebabkan beberapa hal yaitu: harga per kiriman tetap atau konstan dan tidak menganggu penerima.

2.1.1. Karakteristik SMS

Sesuai dengan namanya, SMS berarti layanan pesan pendek, maka besar data yang ditampung oleh SMS sangat terbatas. Untuk satu SMS yang dikirimkan, hanya dapat menampung paling banyak sebesar 140 *byte*. Bila diubah ke dalam bentuk karakter, maka untuk satu SMS hanya dapat berisi paling banyak 160 karakter untuk karakter latin, dan 70 Karakter untuk karakter non-latin seperti karakter Cina maupun Jepang.

Untuk HP yang mampu mengirim SMS lebih dari 160 karakter dalam sekali kirim pada dasarnya bukan berarti SMS memiliki batasan menjadi lebih dari 160 karakter, namun ketika HP mengirimkan SMS yang memiliki karakter lebih dari 160 karakter, HP tersebut akan memecah SMS menjadi bagian-bagian

kecil sebesar 160 karakter, kemudian pada HP penerima bagian SMS yang telah dipotong tersebut digabungkan kembali menjadi satu SMS utuh.

Beberapa karakteristik SMS lainnya, yakni:

a. Pesan SMS dijamin sampai atau tidak sama sekali, selayaknya email, sehingga jika terjadi kegagalan sistem, *timeout*, atau hal lain yang menyebabkan pesan SMS tidak diterima, akan diberikan laporan *(report)* oleh *Short Message Service Center* (SMSC) yang menyatakan bahwa pesan SMS gagal diterima.
b. Berbeda dengan fungsi pemanggilan, sekalipun saat mengirimkan SMS dan HP penerima sedang tidak aktif, bukan berarti pengiriman SMS akan gagal. Namun SMS akan masuk ke antrian terlebih dahulu selama belum *timeout*, SMS akan segera dikirimkan jika HP penerima sudah aktif.
c. Lebar data yang digunakan rendah.
d. SMS yang dikirimkan tidak dapat ditolak oleh HP penerima.

2.1.2. Mekanisme Kerja SMS

Menurut Adi Purnomo (2007), ketika SMS dikirimkan ke suatu nomor tertentu, SMS yang dikirimkan tersebut tidak langsung dikirimkan ke nomor tersebut, namun akan masuk terlebih dahulu ke SMSC operator telepon yang digunakan HP pengirim. SMSC digunakan untuk menjembatani atau menghubungkan antara HP pengirim dan HP penerima.

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

a. Menggunakan terminal baik berupa GSM *modem* atau *handphone*

Cara ini adalah 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 lebih dari satu terminal.

b. Koneksi Langsung ke SMSC

Dengan melakukan koneksi langsung ke SMSC kita dapat mengirim 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. Masing – masing operator GSM menyediakan tipe protokol yang berbeda-beda.

c. Menggunakan *software* bantu

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

Setelah pesan sampai pada SMSC, kemudian akan diteruskan ke *handphone* penerima. Begitu juga sebaliknya. SMSC dapat juga diartikan sebagai perangkat lunak yang terletak di jaringan operator dan mengatur proses-proses, seperti mengatur antrian pesan.

Berikut ini adalah daftar alamat SMSC pada beberapa operator GSM di Indonesia:

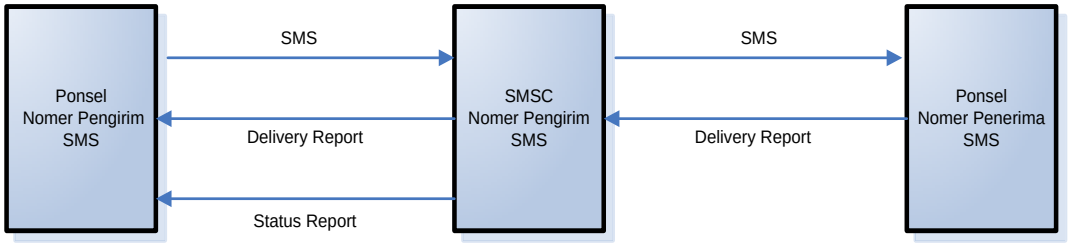
Tabel 2.1. SMSC Operator Selular di Indonesia

Operator GSM	**Nomor SMSC**
Satelindo	62816124
Exelcomindo	62818445009
Telkomsel	6281100000
IM3	62855000000

Sumber: Global Komputer. *SMS*. 21 Maret 2008, para.8, <http://www.globalkomputer.com/ Bahasan/Teknologi/Topik/SMS>

Secara umum, mekanisme kerja pengiriman SMS dibagi menjadi 2, yaitu:

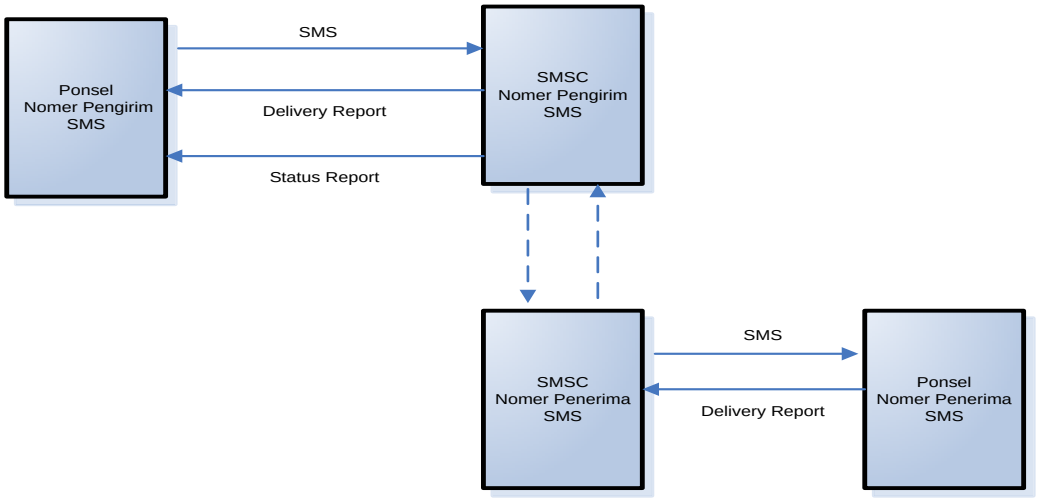
1. Pengiriman SMS dalam satu operator atau sering disebut dengan intra-operator SMS. Gambar 2.1 menjelaskan SMS yang dikirimkan oleh nomor pengirim akan terlebih dulu dimasukkan ke dalam SMSC operator nomor pengirim, kemudian SMSC tersebut akan mengirimkan ke nomor yang dituju secara langsung. Nomor penerima kemudian akan mengirimkan sebuah *delivery report* ke SMSC yang menyatakan bahwa SMS telah diterima. SMSC kemudian meneruskan *delivery report* tersebut ke nomor pengirim SMS, disertai status *report* dari proses pengiriman SMS tersebut.



Gambar 2.1. Mekanisme Pengiriman SMS Intra-operator

Sumber : DevelopersHome Website, *Short Message Service / SMS Tutorial*. 21 Maret 2008. p.6, <http://www.developershome.com/sms/>

2. Pada mekanisme ini, SMS yang dikirimkan akan melalui dua buah SMSC. Gambar 2.2 menjelaskan selain masuk ke SMSC operator pengirim, SMS yang dikirimkan akan diteruskan oleh SMSC operator pengirim ke SMSC operator penerima SMS, kemudian baru diteruskan ke nomor tujuan. *Delivery report* yang dihasilkan pun akan melalui jalur tersebut, agar dapat sampai ke nomor pengirim SMS. Dalam mekanisme ini, terlihat ada sebuah komunikasi tidak langsung antara dua operator berbeda. Komunikasi tersebut dapat berjalan, setelah terjadi sebuah kesepakatan kerjasama antar operator tesebut. Tidak adanya sebuah kesepakatan kerjasama antar operator, dapat menyebabkan SMS yang dikirimkan ke nomor tujuan dengan operator berbeda, tidak sampai pada nomor tujuan tersebut.



Gambar 2.2. Mekanisme Pengiriman SMS Inter-operator

Sumber : DevelopersHome Website, *Short Message Service / SMS Tutorial*. 21 Maret 2008. p.7, <http://www.developershome.com/sms/>

2.2. Algoritma Kompresi Teks

Kompresi ialah proses pengubahan sekumpulan data menjadi suatu bentuk kode untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data. Terdapat beberapa tipe algoritma kompresi teks antara lain: Huffman, *Arithmetic*, *Run-Length-Encoding* (RLE) dan *Lempel-Ziv-Welch* (LZW).

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal menjadi sekumpulan *code number*, metode kompresi teks terbagi menjadi dua kelompok yaitu :

a. Metode Statik, yaitu menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua tahap fase, fase pertama untuk menghitung kemunculan tiap simbol/karakter dan menentukan peta kodenya, dan pada fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan. Contoh: Algoritma Huffman dan Algoritma *Arithmetic*.
b. Metode dinamik (adaptif), yaitu menggunakan peta kode yang dapat berubah dari waktu ke waktu. Metode ini disebut adaptif karena peta kode mampu beradaptasi terhadap perubahan karakteristik isi file selama proses kompresi berlangsung. Contoh: Algoritma LZW dan RLE.

Berdasarkan teknik pengkodean/pengubahan pesan awal atau simbol menjadi sekumpulan *code number*, metode kompresi teks dapat dibagi menjadi dua kategori, yaitu:

a. Metode *Symbolwise*, yaitu menghitung peluang kemunculan dari tiap simbol dalam pesan masukkan, lalu mengkodekan satu simbol dalam satu waktu, di mana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Contoh: Algoritma Huffman
b. Metode *Dictionary*, yaitu menggantikan karakter/fragmen dalam pesan masukkan dengan indeks lokasi dari karakter/fragmen tersebut dalam sebuah kamus. Contoh: Algoritma LZW dan RLE.

2.2.1. Algoritma *Arithmetic Coding*

Algoritma *Arithmetic Coding* merupakan salah satu algoritma kompresi yang bersifat *loseless*, yang berarti jika ada *code number* yang hilang maka *Message* pun akan hilang. Selain itu, *Arithmetic Coding* juga bersifat statik, karena menggunakan tabel probabilitas yang tetap. *Arithmetic Coding* bekerja dengan cara mengubah sebuah *String Message* yang berisi kumpulan karakter menjadi sebuah c*ode number* yang bernilai antara 0 sampai 1. Semua karakter yang mungkin muncul dalam sebuah *String Message* ditentukan probabilitas kemunculannya. Dalam setiap proses *Coding* (pengkodean) ada 2 proses penting yaitu proses *Encoding* dan *Decoding.* Berikut merupakan proses *Encoding* dan *Decoding* dari *Arithmetic Coding.*

2.2.1.1 *Encoding*

Proses ini merupakan proses untuk mengubah sebuah *String Message* menjadi sebuah *code number*. Pada tabel probabilitas karakter yang mungkin muncul dalam sebuah *message*, semua karakter memiliki distribusi probabilitas yang berbeda yaitu dari *Low Range – High Range*. Range ini ditentukan berdasarkan probabilitas karakter tersebut.

Walaupun terdapat karakter yang memiliki probabilitas sama namun *range* probabilitasnya pasti berbeda. Dan *range* semua karakter yang mungkin muncul harus berada dalam range 0 -1.

Prinsip kerja *Encoding* teks pada Algoritma ini adalah sebagai berikut :

1. Nilai *low* dan *high* untuk pertama kali adalah *low* = 0 dan *high* = 1
2. Karakter pertama dicari *current_range = high – low*
3. Cari nilai *high* dan *low* yang baru dengan menggunakan rumus :

   high = *low + current_range x high_range*(karakter pertama)

   low = *low + current_range x low_range*(karakter pertama)
4. Jika tidak terdapat lagi karakter dalam *Message* maka proses berhenti di sini. Namun, jika masih terdapat karakter dalam *Message* maka proses di ulang dari langkah 2 sampai karakter terakhir. Dengan demikian setelah karakter terakhir maka akan diperoleh sebuah *range high* dan *low* yang baru.

Misalnya, telah didefinisikan probabilitas karakter yang mungkin muncul dalam sebuah pesan adalah sebagai berikut :

Karakter	Probabilitas	Range (Low - High)
a	0.2	0.0 - 0.2
b	0.2	0.2 - 0.4
c	0.2	0.4 - 0.6
d	0.2	0.6 - 0.8
e	0.2	0.8 - 1.0

Sebagai contoh agar dapat memahami algoritma ini, misalnya kita akan mengkodekan sebuah *String Message* = "dea". Maka proses encoding akan berlangsung seperti di bawah ini :

En*Code* karakter 'd'

current_range = *high* – *low* = 1 - 0 = 1

high = 0 + (1 x 0.8) = 0.8

low = 0 + (1 x 0.6) = 0.6

En*Code* karakter 'e'

current_range = *high* – *low* = 0.8 – 0.6 = 0.2

high = 0.6 + (0.2 x 1.0) = 0.8

low = 0.6 + (0.2 x 0.8) = 0.76

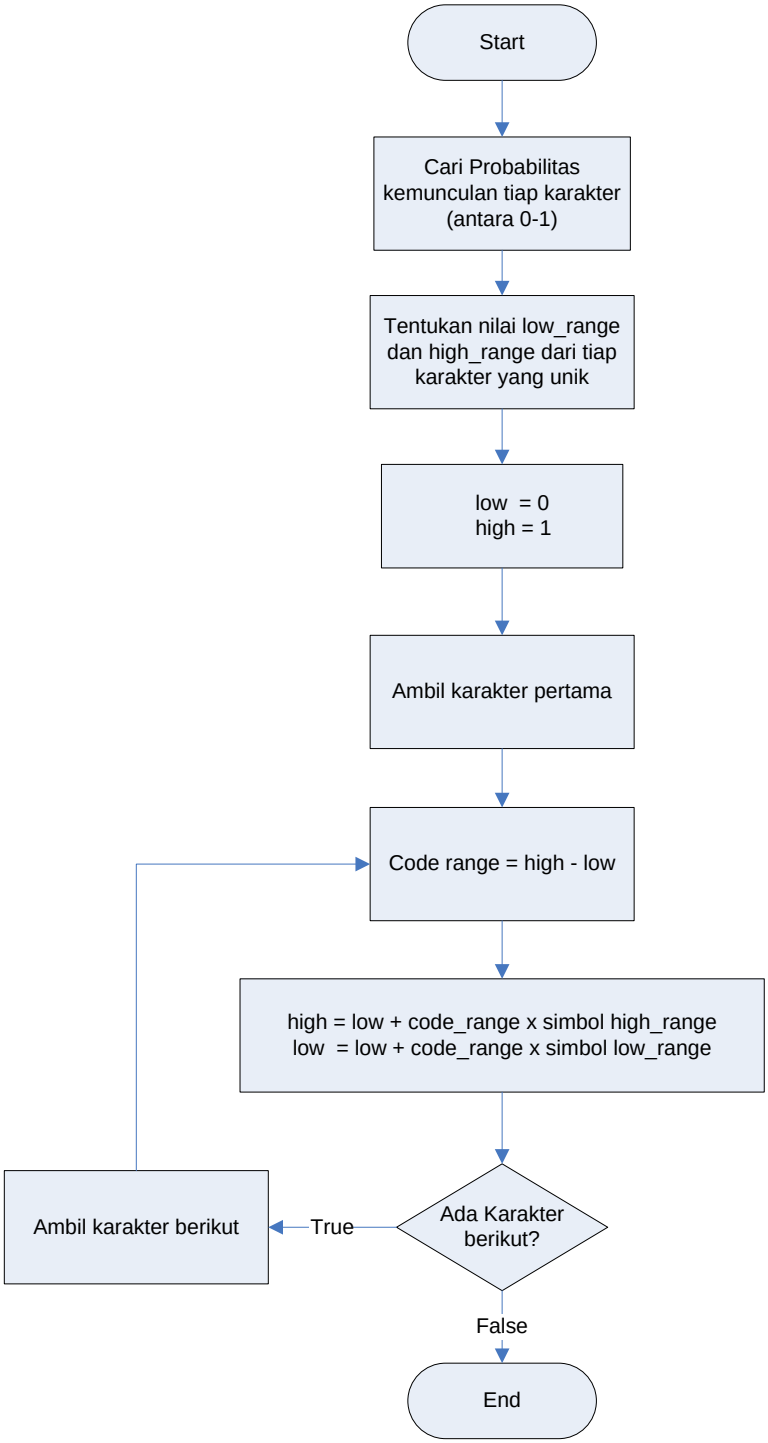
En*Code* karakter 'a'

current_range = *high* – *low* = 0.8 – 0.76 = 0.04

high = 0.76 + (0.04 x 0.2) = 0.768

low = 0.76 + (0.04 x 0.0) = 0.76

Dengan demikian, *range high* – *low* yang diperoleh adalah 0.76 – 0.768. hal ini berarti bahwa *code number* untuk *String Message* "dea" berada dalam range 0.76 - 0.768.



Gambar 2.3. *Flowchart Arithmetic Encoding*

2.2.1.2 *Decoding*

Proses ini merupakan proses untuk mengubah sebuah *Code number* untuk mendapatkan *Message*.

Prinsip kerja algoritma *decoding arithmetic* adalah sebagai berikut:

1. Dari c*ode number* yang diperoleh, dapat langsung ditentukan karakter pertama dari *Message* yang dikirim. Caranya dengan melihat pada tabel probabilitas, di *range* karakter yang manakah *code number* itu berada.
2. Jika sudah mengetahui karakter pertama yang muncul, maka selanjutnya cari *code_range*-nya dengan menggunakan rumus

   code_range = high_range simbol - *low_range* simbol
3. Dapatkan nilai *encoded_number* yang baru dengan menggunakan rumus

   encoded_number = (*encoded_number* - *low_range* symbol) / *code_range*
4. Berdasarkan *encoded_number* kita bisa menentukan karakter baru yang muncul sekarang. Jika data belum selesai, kembali ke langkah 2. Jika sudah berakhir, maka proses selesai.

Misalkan dalam contoh *encoding* di atas, *range code number* yang diperoleh adalah 0.76-0.768. Maka *code number* yang dapat dipakai untuk meng-decode pesan "dea" haruslah angka yang berada dalam *range* tersebut. Dari *range* c*ode number* pesan "dea", dipilih angka yang berada tepat di tengah *range* yaitu angka 0.764. Selanjutnya prosesnya adalah melihat pada tabel probabilitas pada *range* karakter manakah 0.764 berada. Berdasarkan tabel maka karakter pertama yang diperoleh adalah karakter **'d'**. Selanjutnya akan dihitung *Code range* dan *encoded* number dengan rumus di atas.

code_range = *high range* karakter 'd' – *low range* karakter 'd'
= 0.8 - 0.6 = 0.2

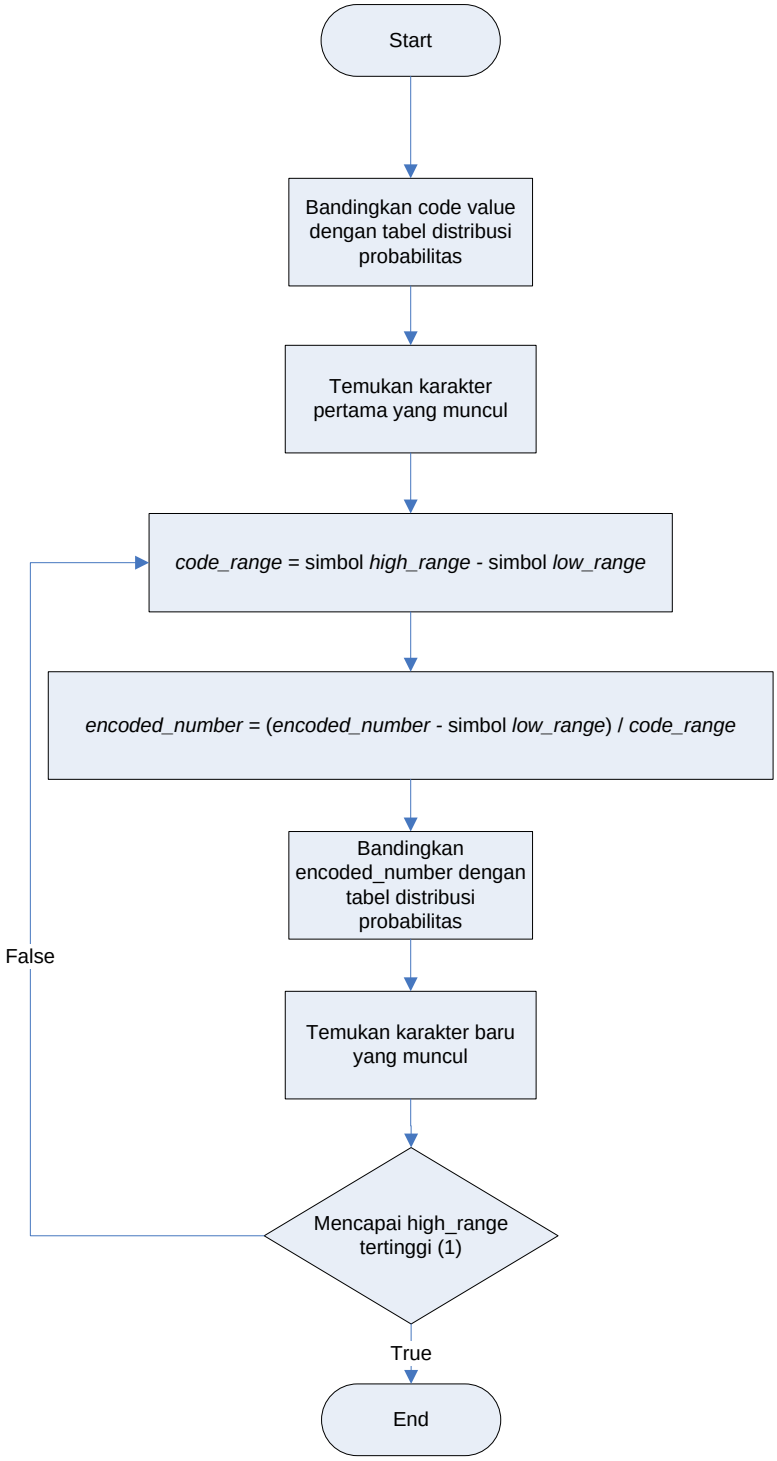
encoded_number = (0.764 – 0.6) / 0.2 = 0.82

Berdasarkan tabel, 0.82 berada pada range probabilitas karakter **'e'.** Selanjutnya perhitungan *Code range* dan *encoded number* yang baru adalah :

code_range = *high range* karakter 'e '– *low range* karakter 'e'
= 1.0 - 0.8 = 0.2

encoded_number = (0.82 – 0.8) / 0.2 = 0.2

Berdasarkan tabel probabilitas, *encoded number* 0.2 berada pada range probabilitas karakter **' a'.** Dengan demikian, pesan yang diperoleh adalah **'dea'.**



Gambar 2.4. *Flowchart Arithmetic Decoding*

2.3. *Java 2 Micro Edition (J2ME)*

Java merupakan nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Java2 merupakan generasi kedua dari Java platform dimana generasi awalnya adalah *Java Development Kit*. Java berdiri di atas sebuah mesin interpreter yang diberi nama *Java Virtual Machine* (JVM). JVM inilah yang akan membaca *bytecode* dalam file .class dari suatu program sebagai representasi langsung program yang berisi bahasa mesin.

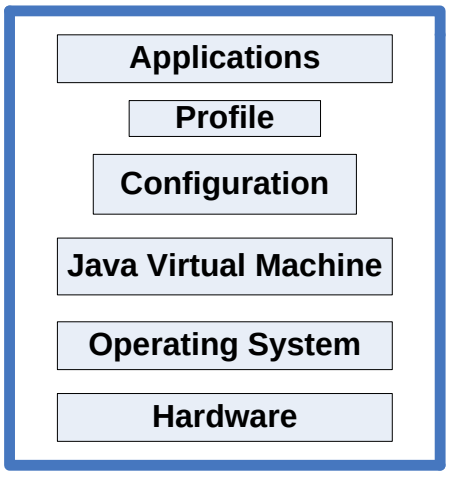
Oleh karena itu, bahasa Java disebut sebagai bahasa pemrograman yang bersifat *portable* atau yang sering disebut dengan *platform-independent* (tidak tergantung pada *platform*). Itulah yang menyebabkan dalam dunia pemrograman java, dikenal adanya istilah '*write once, run everywhere',* yang berarti kode program hanya ditulis sekali, namun dapat dijalankan di bawah *platform* manapun, tanpa harus melakukan perubahan kode program. Sun Microsystems telah mendefinisikan tiga buah edisi dari Java 2, yaitu sebagai berikut:

1. *Java 2 Standard Edition* (J2SE), yang digunakan untuk mengembangkan aplikasi-aplikasi *desktop* dan *applet* (aplikasi Java yang dapat dijalankan di dalam *browser web*).
2. *Java 2 Enterprise Edition* (J2EE), yang merupakan *superset* dari J2SE *yang* memperbolehkan untuk mengembangkan aplikasi-aplikasi berskala besar, yaitu dengan pembuatan aplikasi-aplikasi di sisi *server* dengan mengunakan EJBs (*Enterprise JavaBeans*), aplikasi web dengan menggunakan *Servlet* dan JSP (*Java Server Pages*) dan teknologi lainnya seperti CORBRA (*Common Object Request Broker Architecture)* dan XML (*Extensible Markup Language*).
3. *Java 2 Micro Edition* (J2ME), merupakan *subset* dari J2SE yang digunakan untuk menangani pemrograman di dalam perangkat-perangkat kecil, yang tidak memungkinkan untuk mendukung implementasi dari J2SE secara penuh.

J2ME merupakan sebuah kombinasi yang terbentuk antara sekumpulan *interface* Java yang sering disebut dengan Java API (*Application Programming Interface*) dengan JVM (*Java Virtual Machine*) yang didesain khusus untuk

perangkat dengan ruang memori terbatas. Kombinasi tersebut kemudian digunakan untuk melakukan pembuatan aplikasi-aplikasi yang dapat berjalan pada *mobile device.* Masing-masing dari perusahaan perangkat telah menyediakan JVM dan sekumpulan API yang diperlukan, sehingga tidak perlu dilakukan instalasi JVM dan Java API ke dalam perangkat sehingga *programmer* hanya berkonsentrasi dalam pengembangan aplikasinya dan memasukkannya kedalam perangkat tersebut.

J2ME sendiri pada dasarnya terdiri dari tiga buah bagian, yaitu: konfigurasi, profil, dan paket-paket opsional, seperti yang ditunjukkan pada gambar 2.5. Bagian *Profile* membahas sesuatu yang spesifik untuk sebuah perangkat. Dalam J2ME terdapat 2 buah profile yaitu MIDP dan *Foundation Profile.* Sedangkan *Configuration* mengatur hal-hal tentang kesamaan *device* sehingga dapat dijadikan ukuran kesesuaian antar device. Dalam J2ME didefinisikan dua buah konfigurasi yaitu CLDC (*Connected Limited Device Configuration*) untuk perangkat kecil dan CDC (*Connected Device Configuration*) untuk perangkat yang lebih besar.



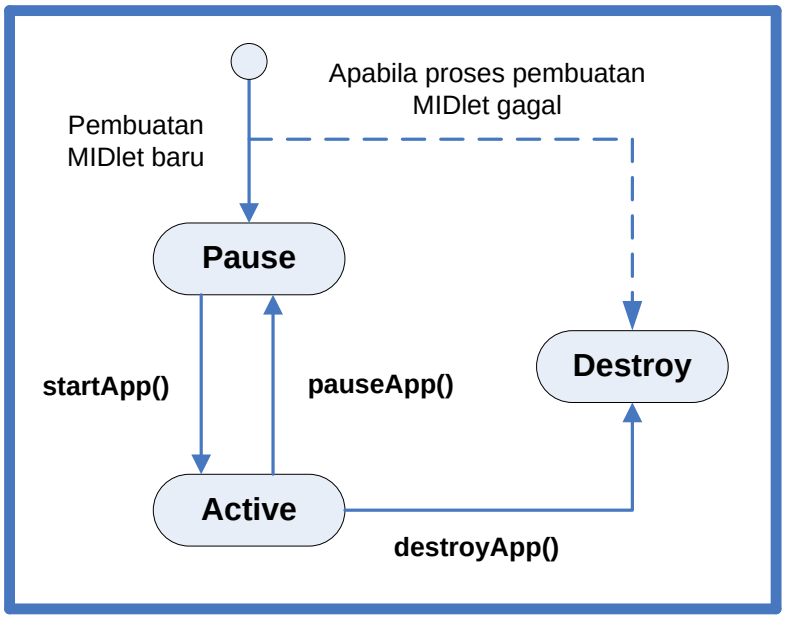
Gambar 2.5. Arsitektur J2ME

Sumber : Raharjo, Budi & Heryanto, Imam. *Tuntunan Pemrograman Java untuk Handphone*. Bandung: Informatika, 2007, p.4 (telah diolah kembali)

2.3.1. Siklus Hidup Aplikasi J2ME

Application Management Software (AMS) merupakan lingkungan tempat sebuah *MIDlet* dapat di-*install*, dijalankan, dihentikan maupun di-*uninstall*. AMS akan membuat *instance* baru dari *MIDlet* dan dapat mengontrol keadaannya, yaitu dengan cara menjalankan (*start*), mengistirahatkan (*pause*) maupun menghentikannya (*destroy*) secara langsung oleh dirinya sendiri.

Pada gambar 2.6 dapat terlihat bahwa terdapat tiga buah *method* yang harus diimplementasikan oleh setiap *MIDlet*. Dengan kata lain, setiap *MIDlet* yang dibuat harus memilik ketiga buah *method* tersebut. Adapun *method-method* tersebut adalah `startApp()`, `pauseApp()`, `destroyApp()`. Setiap *MIDlet* dapat berada dalam salah satu keadaan (*state*) berikut,: *Pause*, *Active*, maupun *Destroyed*. Gambar 2.6 akan mengilustrasikan ketiga buah keadaan tersebut dan pada saat kapan *MIDlet* akan berada dalam keadaan tertentu.



Gambar 2.6. Siklus Hidup *MIDlet*

Sumber : Raharjo, Budi & Heryanto, Imam. *Tuntunan Pemrograman Java untuk Handphone*. Bandung: Informatika, 2007, p.23 (telah diolah kembali)

Tampak pada gambar 2.6 bahwa pada saat pembuatan *MIDlet* baru, mula-mula *MIDlet* akan berada dalam keadaan *Paused*. Apabila proses pembuatan *MIDlet* gagal atau mengakibatkan kesalahan, maka *MIDlet* akan langsung berada pada keadaan *Destroyed.* Namun apabila proses pembuatan *MIDlet* berjalan dengan baik, maka setelah *MIDlet* dijalankan, maka AMS secara otomatis akan mengeksekusi *method* `startApp()` dan hal ini akan mengubah *MIDlet* untuk berada dalam keadaan *Active* dan dapat diubah kembali menjadi keadaan *Paused* melalui pemanggilan *method* `pauseApp()` atau diubah menjadi keadaan *Destroyed* melalui pemanggilan *method* `destroyApp()`**.** Sebagai contoh, pada saat *MIDlet* akan mengalami perubahan keadaan, yaitu dari *Active* menjadi *Destroyed.*

2.3.2. *Connected Limited Device Configuration* (CLDC)

Konfigurasi merupakan bagian yang berisi JVM dan beberapa *library* standar yang digunakan untuk *input, output, security* pada pada *mobile devices* yang support dengan Java.

CLDC adalah sebuah konfigurasi yang terdapat di dalam J2ME untuk alat-alat yang memiliki keterbatasan ruang memori atau RAM (kurang dari 512 KB) dan pada umumnya dioperasikan dengan menggunakan baterai, serta memiliki *bandwith* kecil, contoh alat-alat kecil, seperti telepon selular, PDA dan *pager.*

2.3.3. *Mobile Information Device Profile* (MIDP)

Profil merupakan bagian perluasan dari konfigurasi. Artinya, selain sekumpulan kelas yang terdapat pada konfigurasi, terdapat juga beberapa kelas-kelas spesifik yang didefinisikan lagi dalam profil. Dengan kata lain, profil akan membantu secara fungsional yaitu dengan menyediakan kelas-kelas yang tidak terdapat pada level konfigurasi.

Adapun profil yang sangat popular penggunaannya adalan profil yang disediakan oleh Sun Microsystems, yaitu yang dinamakan dengan MIDP. Beberapa profil yang tersedia untuk kebutuhan-kebutuhan spesifik lainnya:

- *Personal Digital Assistant Profile* (PDAP), yaitu profil untuk PDA yang memperluas fungsi-fungsi pada konfigurasi CLDC dan digunakan khusus

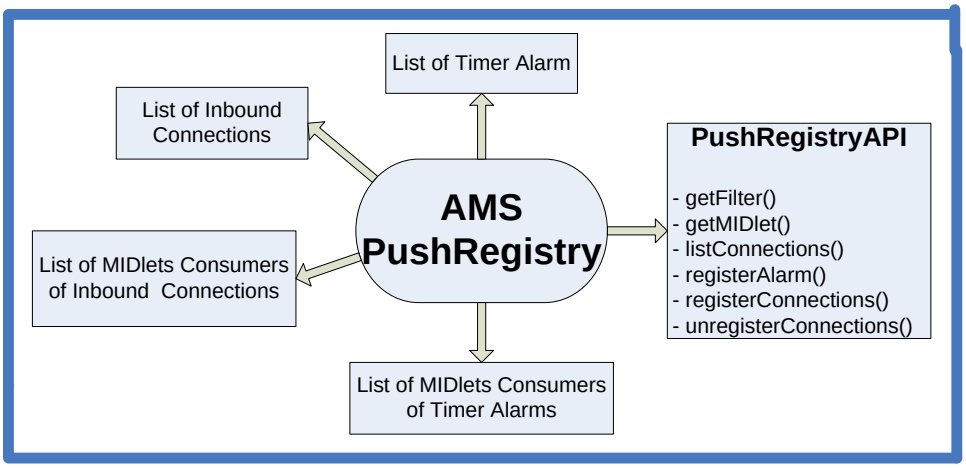
untuk menambahkan kemampuan-kemampuan lebih apabila dibandingkan dengan penggunaan MIDP

- *Foundation Profile,* yaitu profil yang digunakan untuk konfgurasi CDC. Profil ini menambahkan beberapa kelas dari J2SE ke dalam konfigurasi CDC, dan berperan juga dalam pondasi untuk membentuk profil baru lainnya.
- *Personal Profile,* yaitu profil yang mendefinisikan ulang personal Java sebagai profil yang dapat digunakan sebagai profil dalam J2ME. Profil ini merupakan hasil perluasan dari *Foundation profile*
- *Remote Method Invocation* (RMI), yaitu profil yang menambakan dukungan RMI ke dalam konfigurasi CDC.

2.3.4. Push Registry

Menurut Ortiz (2003), *Push Registry* adalah suatu mekanisme dalam *MIDlet* untuk menghidupkan aplikasi *MIDlet* secara otomatis tanpa ada campur tangan dari pengguna, dengan mengirimkan sinyal tertentu ke *handphone* sehingga aplikasi di *handphone* bisa hidup. Sinyal yang dikirimkan bisa berupa SMS, socket atau datagram.

Push Registry terletak di dalam klas javax.microedition.io.PushRegistry pada MIDP 2.0. gambar 2.7 menjelaskan elemen-elemen *Push Registry*:



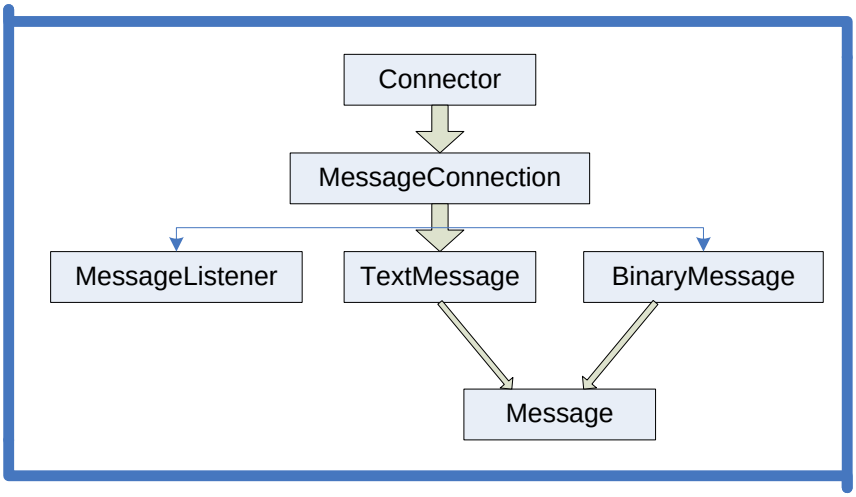
Gambar 2.7. Elemen-elemen *Push Registry*

Sumber : Ortiz, Enrique. *The MIDP 2.0 Push Registry*, Januari 2003, 24 April 2008. p.1, <http://developers.sun.com/mobility/midp/articles/pushreg/>

Untuk mengimplementasikan *Push Registry* pada aplikasi *MIDlet*, pada file *java application descriptor* (jad) dilakukan penambahan script untuk mendaftarkan nomor port yang akan di-*listen* oleh AMS, *MIDlet*-Push:sms://:5000 adalah cara meregister AMS untuk mendengarkan koneksi SMS pada port 5000 dan apabila ada SMS yang masuk pada port tersebut maka SMS tidak akan masuk pada aplikasi SMS bawaan *handphone* melainkan AMS akan menghidupkan *MIDlet* secara otomatis dan mengirimkan isi SMS tersebut untuk diproses oleh aplikasi *MIDlet*.

2.3.5. *Wireless Messaging API* (WMA)

WMA adalah paket opsional yang terdapat pada J2ME, yang mengizinkan pengembang untuk mengembangkan aplikasi-aplikasi yang mampu melakukan pengiriman dan penerimaan pesan SMS dengan meng-*import* kelas pada paket javax.wireless.messaging.



Gambar 2.8. *Interface* pada paket WMA

Sumber : Raharjo, Budi & Heryanto, Imam. *Tuntunan Pemrograman Java untuk Handphone*. Bandung: Informatika, 2007, p.142 (telah diolah kembali)

Pada gambar 2.8 dapat dilihat bahwa dalam proses pengiriman dan penerimaan pesan SMS, terdapat tiga buah *interface* antara lain: *TextMessage*, *BinaryMessage*, dan *MessageConnection* yang mendefinisikan *method* umum untuk mengeset alamat penerima dan juga mendapatkan waktu pesan.

Berikut ini deklarasi dari *method-method* yang terdapat dalam *interface message*.

```
String getAddress()                          // mengambil alamat pengirim
void setAddress (String address)             // mengeset alamat tujuan
Date getTimeStamp                            // mengambil tanggal pengiriman
String getPayLoadText()                      // mengambil isi pesan teks
void setPayloadText (String body)            // menampung pesan teks
byte[] getPayloadData()                      // mengambil isi pesan biner
void setPayloadData(byte[] content)          // menanmpuk pesan biner
```

Gambar 2.9. *Syntax Interface Message*

Sumber : Raharjo, Budi & Heryanto, Imam. *Tuntunan Pemrograman Java untuk Handphone*. Bandung: Informatika, 2007, p.140 (telah diolah kembali)

Inti dari kelas WMA berada pada *interface MessageConnection*, yang merepresentasikan sebuah koneksi jaringan untuk memperoleh proses pengiriman maupun penerimaan pesan dengan cara melewatkan URL tertentu ke dalam `method Connector.open().`

Berikut ini aturan penulisan URL tertentu yang diizinkan di dalam WMA:

- **sms://no_telepon,** MessageConnection akan mengirimkan pesan ke nomor telepon tujuan. Pesan akan terkirim ke *inbox* SMS dari *device* tujuan. Dengan demikian, pesan secara otomatis akan diterima oleh aplikasi yang telah disediakan oleh *device* bersangkutan, bukan oleh aplikasi penerima SMS yang akan kita kembangkan sendiri.
- **sms://no_telepon:port,** MessageConnection akan mengirimkan pesan ke no telepon tujuan untuk *port* yang telah ditentukan. Di sini pesan tidak akan terkirim ke *inbox* SMS dari *device* bersangkutan melainkan akan dikirimkan ke suatau *MIDlet* pada *device* penerimayang bertugas mendengarkan *port* tertentu.
- **sms://:port,** MessageConnection akan mendengarkan port yang ditentukan. *MIDlet* SMS yang berada di *client* berperan sebagai *server* pada port tertentu. Pesan akan terkirim melalui port tersebut. Koneksi jenis ini dinamakan dengan koneksi mode server.

Interface MessageConnection mendeklarasikan beberapa buah *method* untuk keperluan pengiriman dan penerimaan pesan, yaitu sebagai berikut

```
Message newMessage(String type)                  // membuat tipe pesan baru
int numOfSegments(Message msg)                   // mengambil jumlah sms
Message receive()                                // menerima sms
void send(Message msg)                           // mengirim sms
void setMessageListener(MessageListener l) // mendengarkan portsms
```

Gambar 2.10. *Syntax Interface MessageConnection*

Sumber : Raharjo, Budi & Heryanto, Imam. *Tuntunan Pemrograman Java untuk Handphone*. Bandung: Informatika, 2007, p.141 (telah diolah kembali)

Parameter type yang terdapat pada *method* newMessage() dapat berupa TEXT_MESSAGE atau BINARY_MESSAGE. MessageConnection juga dapat memiliki sebuah objek *listener. MIDlet* yang memiliki objek *listener* harus mengimplementasikan *interface MessageListener*.

2.4 *Thread*

Sebuah *thread* adalah satuan dasar di mana sistem operasi mengalokasikan waktu prosesornya. Setiap *thread* menangani *exception handlers*, sebuah prioritas penjadwalan, dan satu set struktur di mana sistem akan menggunakannya untuk menyimpan konteks *thread* sampai ia terjadwal. Yang termasuk di dalam konteks *thread* adalah mesin, *register* dan *stack* yang berada dalam alamat prosesnya.

Sebuah sistem operasi *multitasking* membagi waktu *prosesor* untuk proses atau *thread* yang membutuhkannya. Sistem mengalokasikan potongan waktu *prosesor* ke setiap *thread* yang dieksekusi. *Thread* yang sedang dieksekusi akan ditangguhkan ketika potongan waktunya habis, kemudian *thread* lain ganti dijalankan. Status *thread* ketika terjadi pergiliran disimpan di dalam antrian. Lama potongan waktu prosesor bergantung pada sistem operasi yang bersangkutan dan juga prosesornya sendiri. Karena tiap potongan waktu itu begitu singkatnya maka *thread* ganda terlihat dieksekusi pada waktu yang bersamaan.

Pada umumnya *thread* digunakan untuk aplikasi yang :

- Melakukan operasi yang membutuhkan waktu cukup lama.
- Membedakan *task* yang mempunyai prioritas berbeda-beda.
- Mempertahankan antarmuka pemakai supaya tetap *responsif*, selama melakukan proses *background*.

2.4.1. Single Thread

Setiap program java memiliki setidaknya sebuah thread, yaitu main yang merupakan *single-thread* tersendiri di JVM. Java juga menyediakan perintah untuk membuat dan memodifikasi *thread* tambahan sesuai Kebutuhan program.

Salah satu cara membuat thread secara eksplisit yaitu dengan membuat objek baru dari class yang telah *extends* class *Thread*. Cara lain adalah dengan *override method run* dari *interface Runnable*. Sebuah objek yang berasal dari subkelas *Thread* dapat dijalankan sebagai kontrol *thread* yang terpisah dalam JVM. Membuat objek dari *class Thread* tidak akan membuat *thread* baru. Akan tetapi dengan `method start()` akan terbentuk *thread* baru.

Memanggil `method start()` untuk objek baru akan mengakibatkan 2 hal yaitu:

- Mengalokasikan memori dan menginisialisasi sebuah *thread* baru dalam JVM
- Memanggil *method* run, membuat *thread* dapat dijalankan oleh JVM (Catatan: *Method* run dijalankan jika *method* `start()` dipanggil. Memanggil *method* run secara langsung hanya menghasilkan *single-thread* tambahan selain main. pembuatan *thread* dengan membuat objek baru dari *class* yang *extends* class *Thread*

```
public class TestThread1 {
    public static void main (String[] args) {
        BuatThread1 b = new BuatThread1();
        for(int i = 0; i < angka; i++) {
            b.start();
        }
    }
}

class BuatThread1 extends Thread {
    public void run() {
        try {
            System.out.println("Thread baru dibuat.");
        }
        catch (InterruptedException e) {
        }
    }
}
```

Gambar 2.11. *Syntax* Membuat Thread dalam Java

2.4.2. Multi Thread

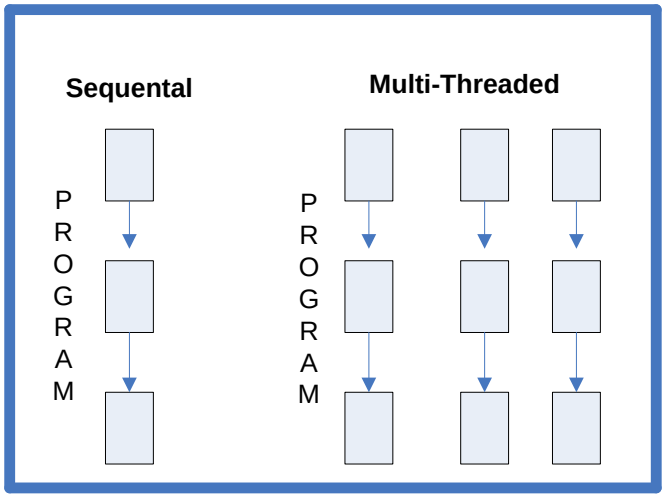
Dahulu komputer hanya mempunyai satu CPU tunggal, dan hanya mampu mengeksekusi satu program pada satu waktu. Kemudian datang *multitasking* yang berarti bahwa komputer dapat mengeksekusi banyak program (disebut juga task atau proses) pada waktu yang sama. Sebenarnya tidak benar-benar "pada waktu yang sama". Satu CPU tersebut digilir (*share*) antar beberapa program. Sistem operasi mengatur giliran berjalannya program, mengeksekusi setiap program tersebut selama waktu yang sangat singkat sebelum pindah giliran.

Bersama dengan *multitasking* hadir tantangan baru bagi pengembang perangkat lunak. Suatu program tidak dapat lagi menguasai semua waktu CPU yang tersedia, tidak juga terhadap memory atau sumber daya komputer yang lain. Program "yang baik" akan melepas semua sumber daya yang sudah tidak digunakannya, sehingga dapat digunakan oleh program-program lain. Kemudian datang *multithreading* yang berarti bahwa kita dapat mempunyai banyak *thread* eksekusi di dalam program yang sama. Ketika kita mempunyai banyak *thread* yang mengeksekusi program yang sama, seolah-olah kita mempunyai banyak CPU yang mengeksekusi program tersebut.

.

Saat menggunakan *multithreading*, satu *thread* memproses antarmuka sementara *thread* lain melakukan kalkulasi-kalkulasi intensif atau memproses di latar. Bahasa pemrograman Java memfasilitasi *multithreading*, sehingga para pengembang program dapat dengan mudah menggunakan kemudahan ini.

Cara paling mudah untuk membuat proses latar yang dapat berproses di *thread*-nya sendiri dengan datanya sendiri adalah membuat suatu obyek khusus untuk proses latar. Tujuan dilakukannya hal ini adalah baik, sepanjang dapat menyederhanakan pembuatan aplikasi *multithread*. Jika *background thread* melakukan proses di dalam obyeknya sendiri, maka ia dapat memakai variabel instan dari obyek tersebut tanpa khawatir bahwa mereka akan dipakai oleh *thread* yang lain. Perbedaan proses yang dilakukan antara *single-thread* dengan *multithread* dapat dilihat pada gambar 2.12.



Gambar 2.12. Perbedaan antara *Single Thread* dan *Multi Thread*

Mulithreading memungkinkan *thread* berjalan di dalam program yang sama dan karena itu dapat membaca dan menulis memory yang sama secara simultan. Ini dapat mengakibatkan *error* tidak terlihat di dalam program *singlethreaded.* Beberapa *error* ini dapat tidak terlihat pada mesin CPU tunggal, karena dua *thread* tidak pernah benar-benar dieksekusi secara simultan. Komputer modern, datang dengan CPU *multi-core*. Ini berarti bahwa *thread-thread* terpisah dapat dieksekusi oleh *core* terpisah secara simultan.