BAB II

DASAR TEORI

Dalam bab ini akan dijelaskan tentang Java secara umum, JME (*Java Micro Edition*) sebagai bahasa pemrograman yang akan digunakan dalam membangun aplikasi, pemrograman berorientasi objek, Leitner System yang merupakan cikal bakal dari banyak aplikasi *flashcard* (Wozniak, 2006), gambaran umum aplikasi *flashcard*, NetBeans sebagai IDE, serta *Unified Modelling Language*.

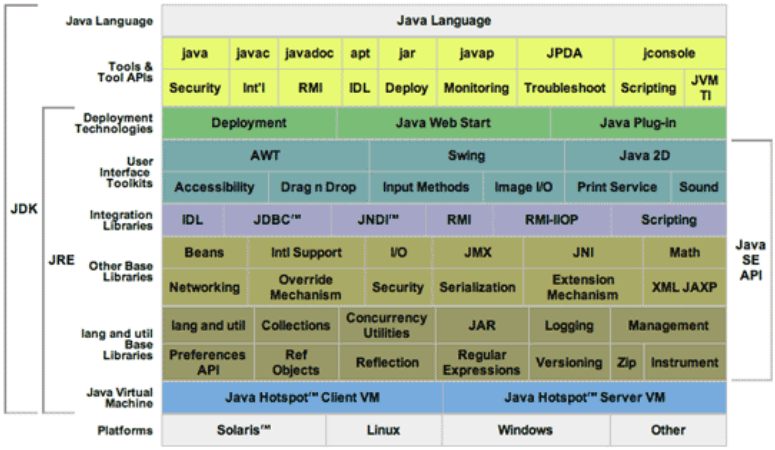
2.1 Java

Java adalah bahasa pemrograman yang disusun oleh James Gosling yang dibantu rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank, dan Mike Sheridan di suatu perusahaan perangkat lunak yang bernama Sun Microsystems pada tahun 1991. Bahasa pemrograman ini mula-mula diinisialisasi dengan nama "Oak", namun pada tahun 1995 diganti namanya menjadi "Java" (Raharjo, 2007).

Java menurut definisi dari Sun adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Java2 adalah generasi kedua dari Java *platform* (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 berkas *.class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin. Oleh karena itu bahasa Java disebut

sebagai bahasa pemrograman *portable* karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM (Shalahuddin, 2006).

Pada tahun 2006, Sun menyederhanakan penamaan *platform* untuk mencerminkan kematangan, kestabilan, skalabilitas, dan keamanan dalam Java *platform*. Sun menghilangkan "2" dari nama *platform* dan meniadakan *dot number*, angka yang mengikuti tanda titik. Pada rilis saat ini, nama *platform* berubah dari J2SE™ menjadi Java™ SE atau Java Platform, Standard Edition 6. Dua macam penomoran versi (1.6.0 dan 6) digunakan untuk mengidentifikasi rilis saat ini. Versi 6 adalah versi produk, sedangkan 1.6.0 adalah versi pengembang (http://java.sun.com, 12 Agustus 2008). Beberapa referensi yang dipublikasikan setelah tahun 2006 masih menuliskan J2SE daripada JSE. Gambar 2.1 menjelaskan tentang arsitektur di Java.

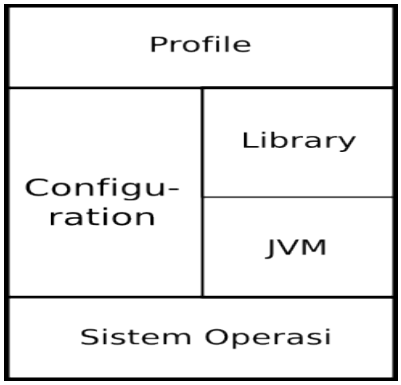


Gambar 2.1 Arsitektur Java (http://java.sun.com, 2008)

2.2 *Java Micro Edition*

Java Micro Edition 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 alat, yaitu JVM dengan ruang yang terbatas (Raharjo, 2007). Seperti halnya penamaan pada JSE, saat ini masih terdapat beberapa referensi yang menuliskan J2ME daripada JME. Penulis akan lebih sering menggunakan istilah *Java Micro Edition* atau JME daripada J2ME pada tulisan ini, walaupun terkadang dalam referensi disebutkan cara penamaan yang lama.

J2ME adalah bagian dari J2SE, karena itu tidak semua *library* yang ada pada J2SE dapat digunakan pada J2ME. Tetapi J2ME mempunyai beberapa *library* khusus yang tidak dimiliki J2SE (Shalahuddin, 2006). Gambar 2.2 menunjukkan arsitektur JME.



Gambar 2.2 Arsitektur JME
(Shalahuddin, 2006)

2.2.1 *Configuration* dan *Profile* pada JME

Configuration mendefinisikan minimum *Java Libraries* dan kemampuan yang dipunyai oleh para *developer* JME. Artinya pada semua *mobile device* yang

Java enabled maka akan ditemui *configuration* yang sama, sehingga *configuration* memastikan portabilitas antar piranti (Wiryasantika, 2003).

Saat ini terdapat dua macam *configuration* untuk JME (Wicaksono, 2002), yaitu :

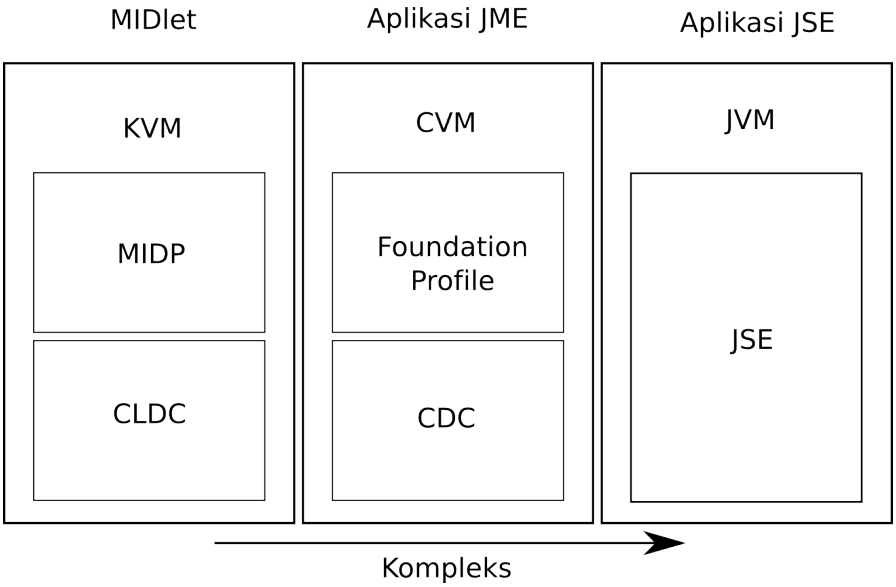
1. CLDC (*Connected Limited Device Configuration*)

   Configuration jenis ini digunakan untuk aplikasi Java pada *handphone*, *Pocket PC*, dan *two way pages*. Perangkat di atas umumnya mempunyai antar muka yang sederhana, ukuran memori lebih kecil (128 kilobytes – 1 megabyte) CPU 16 bit atau 32 bit dan *bandwith* yang kecil.

2. CDC (*Connected Device Configuration*)

   Configuration jenis ini digunakan untuk aplikasi Java seperti internal TV, Nokia Communication dan TV Mobile, umumnya perangkat tersebut mempunyai ukuran memori 2 sampai 16 megabyte CPU 32 bit atau lebih dan *bandwith* yang cukup tinggi.

Profile berbeda dengan *configuration*, *profile* membahas sesuatu yang spesifik untuk sebuah perangkat. Sebagai contoh misalnya, sebuah sepeda dengan merk tertentu tentu mempunyai ciri spesifik dengan sepeda lain. Dalam JME terdapat dua buah *profile* yaitu MIDP dan *Foundation Profile* (Shalahuddin, 2006). Gambar 2.3 menunjukkan hubungan antara JME dan JSE.



Gambar 2.3. Hubungan JME dan JSE (Shalahuddin, 2006)

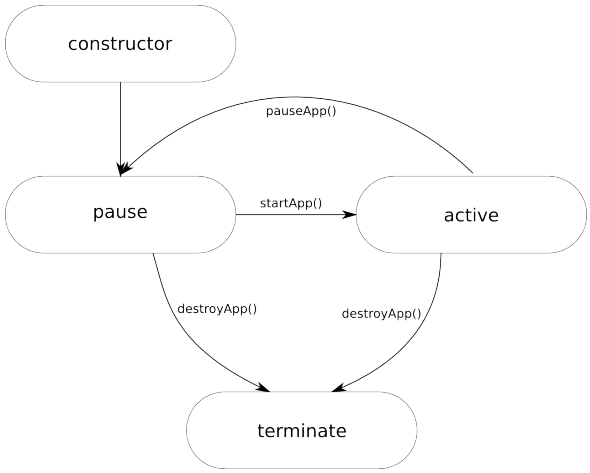
2.2.2 MIDP

MIDP atau *Mobile Information Device Profile* adalah spesifikasi untuk sebuah profil J2ME. MIDP memiliki lapisan di atas CLDC, API tambahan untuk daur hidup aplikasi, antarmuka, jaringan, dan penyimpanan persisten. Pada saat ini terdapat MIDP 1.0 dan MIDP 2.0. Fitur tambahan MIDP 2.0 dibanding MIDP 1.0 adalah API untuk multimedia. Pada MIDP 2.0 terdapat dukungan memainkan *tone, tone sequence*, dan file WAV, walaupun tanpa adanya *Mobile Media API* (MMAPI) (Shalahuddin, 2006).

2.2.3 MIDlet

MIDlet adalah aplikasi yang ditulis untuk MIDP. Aplikasi MIDlet adalah bagian dari kelas javax.microedition.midlet.MIDlet yang didefinisikan pada MIDP. MIDlet berupa sebuah kelas abstrak yang merupakan sub kelas dari bentuk dasar aplikasi sehingga antarmuka antara aplikasi J2ME dan aplikasi manajemen pada

perangkat dapat terbentuk (Shalahuddin, 2006). Daur hidup MIDlet ditunjukkan oleh gambar 2.4.



Gambar 2.4. Daur hidup MIDlet
(Shalahuddin, 2006)

2.3 Pemrograman Berorientasi Objek

2.3.1 Objek

Objek adalah benda, baik yang berwujud nyata maupun tidak nyata. Dalam kosep pemrograman berorientasi objek, objek adalah unit terkecil pemrograman yang masih memiliki data (sifat karakteristik) dan fungsi (Hermawan, 2004).

2.3.2 Kelas

Kelas adalah wadah yang berisi abstraksi (pemodelan) dari suatu objek (benda), yang mendeskripsikan data dan fungsi yang dimiliki oleh objek tersebut. Karena merupakan wadah dari objek, maka kelas dibuat sebelum objek (Hermawan, 2004).

2.3.3 Instans

Objek dapat dikatakan juga sebagai instans dari suatu kelas. Maksud dari instans dalam konsep pemrograman adalah merupakan perwujudan dari suatu kelas. Sedangkan proses pembentukan objek dari suatu kelas disebut instansiasi (Hermawan, 2004).

2.3.4 Pewarisan

Menurut Yulianto (2004), *inheritance* atau pewarisan memungkinkan untuk menciptakan sebuah kelas dari kelas yang sudah ada. Ketika sebuah kelas diturunkan dari kelas yang sudah ada maka kelas tersebut mewarisi semua *method* dan *field* dari kelas orang tuanya. Kelas anak juga mewarisi bentuk dari kelas orang tua. Artinya objek dari kelas anak dapat dikenali sebagai objek dari kelas orang tua.

Mewarisi berarti memiliki method dan fungsi dari kelas induk. Memiliki tidak sama dengan kemampuan mengakses *method* dan *field* yang diwarisi. Hak akses ditentukan oleh *access specifier* (public, protected, private). Kelas anak hanya dapat mengakses *method* dan *field* yang public dan protected (Yulianto, 2004).

2.3.5 Polimorfisme

Dalam konsep pemrograman polimorfisme digunakan untuk menyatakan satu nama yang merujuk ke beberapa fungsi yang berbeda. Java mengenal dua jenis polimorfisme yaitu *method overloading* dan *method overriding* (Yulianto, 2004).

2.4 *Leitner System*

Menurut Interfour (2006), mempelajari banyak informasi sekaligus dalam waktu yang relatif singkat terkadang dapat menyiksa dan membuat frustasi. Karena manusia cenderung mempelajari hal yang sama secara berulang-ulang, maka kebosanan tidak dapat dihindarkan. Oleh sebab itu perlu adanya pembelajaran yang selektif, yaitu pembelajaran yang fokus pada hal-hal yang sulit diingat.

Sebastian Leitner adalah seorang psikolog Eropa yang menemukan sistem kotak yang disebut *Leitner System* pada tahun 1960-an. Tujuannya adalah untuk mempermudah proses belajar dan mengingat kosakata untuk memori jangka pendek maupun jangka panjang (MemoryLifter, 2008). Ide Leitner yang sederhana membuat efek yang sangat besar pada kinerja pembelajaran (Interfour, 2006). *Card* atau kartu pada sistem milik Leitner sebenarnya adalah kartu biasa yang di salah satu sisinya berisi pertanyaan sedangkan di sisi yang lain terdapat jawaban yang bersesuaian.

Menurut Interfour (2006), *Leitner System* terdiri dari sebuah kotak atau *box* yang dibagi menjadi beberapa bagian atau kompartemen. Kompartemen-kompartemen tersebut diisi dengan *flashcards* atau kartu yang disusun berdasar tingkat kesulitannya. *Flashcards* atau kumpulan kartu-kartu tersebut dapat berpindah kompartemen sesuai dengan tingkat pengetahuan kartu yang bersangkutan. Ketika sebuah *card* atau kartu dapat dijawab dengan baik, maka kartu tersebut dipromosikan ke kompartemen setelahnya. Sedangkan bila kartu tersebut gagal dijawab dengan baik maka kartu akan berpindah ke kompartemen

paling depan. Cara kerja *Leitner system* terlihat pada gambar 2.5.

Gambar 2.5. Cara kerja *Leitner System* (Interfour, 2006)

Dengan demikian, cara *Leitner System* akan membuat penggunanya menghafal kartu yang sulit diingat (kompartemen paling depan) lebih sering daripada kartu yang mudah diingat (kompartemen paling belakang) sampai semua kartu dapat dihafal dengan baik (kondisi semua kartu terletak di kompartemen paling akhir).

Menurut Interfour (2006), keuntungan dari *learning cardfiles* milik Leitner adalah sebagai berikut

1. Belajar secara selektif.

   Masing-masing kotak dalam Leitner System menunjukkan tingkat kesulitan, sehingga pengguna bebas memilih antara belajar *item-item* yang dihafal dengan baik atau mempelajari hal-hal yang sulit.

2. Hasil yang dapat terlihat dengan segera membuat diri pengguna termotivasi.

   Status yang menunjukkan seberapa besar pengetahuan yang telah didapat pengguna ditunjukkan oleh seberapa banyak kartu atau *flashcard* yang terdapat dalam setiap kotak. Kecurangan hampir tidak mungkin terjadi.

3. Kegagalan dalam mengingat dapat diatasi.

   Setiap sebuah *flashcard* atau kartu yang gagal dijawab dengan baik

akan masuk ke kompartemen pertama. Ini membuat kartu tersebut dipelajari berulang, dan secara perlahan-lahan akan sukses dihafal.

4. Tanggapan secara langsung.

   Respon dari pengguna diperiksa secara langsung, sehingga pengguna tidak perlu menunggu lama sampai menjawab dengan benar untuk mengetahui apakah jawaban pengguna benar atau tidak.

5. Mengubah urutan.

   Flashcards yang secara selektif dipilih dari kompartemen tertentu dapat diatur urutannya.

6. Latihan yang tidak menyebabkan depresi.

   Mempermasalahkan opini publik merupakan salah satu penyebab *stress*. Hal ini berlaku untuk anak-anak dan orang dewasa. Dengan menggunakan *Leitner system*, pengguna dapat mengatur sendiri kecepatan belajar, serta lebih berkonsentrasi pada materi belajar daripada faktor eksternal.

7. Mendukung keahlian belajar-mandiri.

   Dengan *Leitner system*, pengguna atau siswa dihadapkan pada kenyataan bahwa tidak ada guru yang dapat ditanyai ketika mereka merasa kesulitan dalam menjawab pertanyaan. Kesuksesan dapat membantu rasa percaya diri terhadap kemampuan sendiri.

2.5 Aplikasi *Flashcard*

2.5.1 SuperMemo

2.5.1.1 *Space repetition*

Menurut Wozniak (2006), pada tahun 1901 seorang psikolog Amerika bernama William James menyimpulkan bahwa review pada materi belajar seharusnya dilakukan dalam waktu yang berjangka. Kesimpulan yang sama juga diajukan Hermann Ebbinghaus dan Jost. Pada tahun 1932, C.A. Mace memperkenalkan metode belajar yang efisien dalam bukunya yang berjudul "The Psychology of Study". Ia memperkenalkan istilah *active rehearsal* dan *repetitive revisions* yang menyebutkan bahwa proses belajar seharusnya berjangka menurut interval waktu yang meningkat, misalnya "interval 1 hari, 2 hari, 4 hari, dan seterusnya". Pendekatan ini diadopsi oleh beberapa penulis lain (termasuk Tony Buzan), namun tidak pernah diaplikasikan pada lingkungan studi dan penelitian. Barulah sekitar 50 tahun kemudian, ide ini diimplementasikan pada komputer sehingga menciptakan jalan yang lebih lebar bagi publik untuk melihat keuntungan *spaced repetition*.

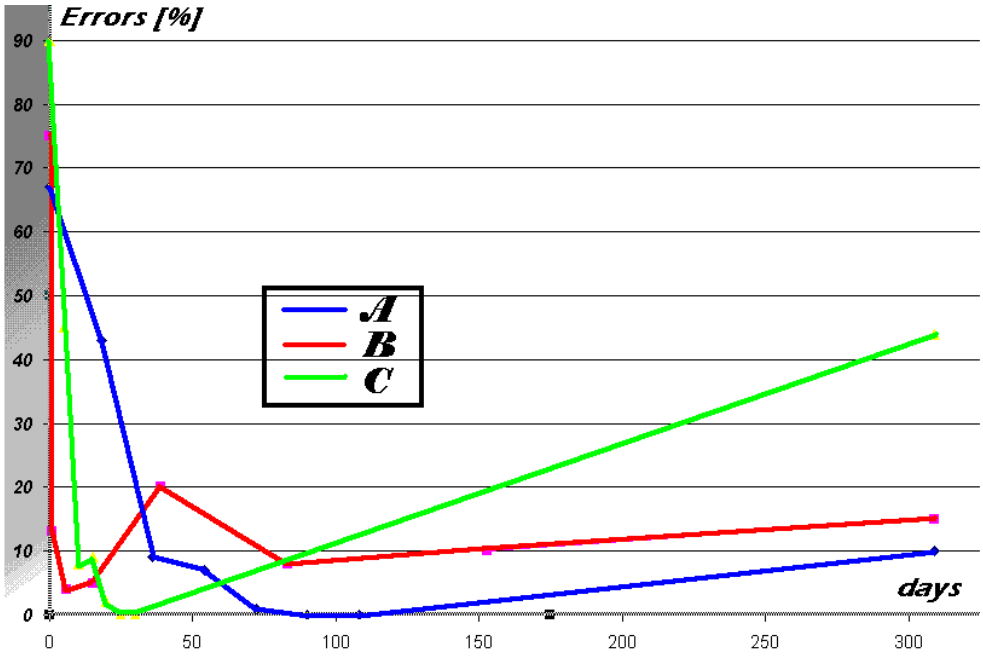
Spaced repetition (perulangan berjangka) adalah teknik optimal dalam proses belajar dengan cara menghitung jangka waktu atau interval yang memisahkan proses mengingat (mengambil informasi dari memori) dari pengetahuan tertentu (Wozniak, 2006). Piotr Wozniak yang ketika itu adalah seorang mahasiswa ilmu komputer merupakan orang yang pertama kali menulis aplikasi *flashcard*, program komputer yang menerapkan prinsip *spaced repetition*. Dalam waktu 16 malam, ia berhasil membuat SuperMemo 1.0 untuk MS DOS

dengan Borland's Turbo Pascal 3. Sejak 13 Desember 1987, Wozniak mempelajari tentang Biologi Manusia dengan menggunakan SuperMemo 1.0. Beberapa tahun setelahnya, SuperMemo menjadi aplikasi yang sifatnya komersial (Wozniak, 2006).

2.5.1.2 Metode penjadwalan

Ada beberapa metode yang digunakan dalam penjadwalan *item* yang akan dipelajari. Pada aplikasi SuperMemo, algoritma yang digunakan adalah algoritma SM (*SM-algorithm*). Sedangkan pada Trivial, sebuah aplikasi *flashcard* yang ditulis dengan bahasa C#, digunakan algoritma FSM atau *Finite State Machine* (Rachmatullah, 2008). Algoritma untuk aplikasi SuperMemo terus menerus mengalami perkembangan mulai tahun 1985 hingga saat ini, mulai dari SM-0 hingga yang terakhir SM-11.

Menurut Wozniak (1990), pernyataan bahwa perulangan yang berjangka secara meningkat (*increasing inter-repetition intervals*) tidak selamanya lebih efektif bila dibandingkan perulangan dengan jangka waktu antar review yang sama. Hal ini dibuktikan dengan percobaan yang dilakukan Wozniak sejak akhir Januari tahun 1985 hingga awal Agustus 1986 terhadap tiga macam kelompok pengetahuan yang diperlakukan berbeda. Kelompok A diulang dengan interval waktu yang sama (18 hari). Kelompok B diulang secara berjangka dan meningkat mulai 1 hari hingga 70 hari. Kelompok C diulang dengan jarak waktu yang sama, namun dengan masa yang relatif lebih singkat dari kelompok A, yakni setiap 5 hari. Hasilnya ditunjukkan oleh gambar 2.6.



Gambar 2.6. Hasil percobaan Wozniak pada 3 kelompok (Wozniak, 1990)

Gambar 2.6 menunjukkan bahwa hasil yang didapat dengan menggunakan perulangan dengan jangka waktu yang meningkat tidak lebih baik dari perulangan yang memakai jangka waktu yang tetap. Prinsip dalam menghitung interval waktu yang digunakan dalam proses belajar merujuk pada istilah *optimum repetition spacing principle* (Wozniak, 1990). Permasalahannya adalah semakin lama jangka waktu untuk sebuah *item* agar dapat diulang membuat semakin sulitnya informasi bersangkutan diakses kembali (Wozniak, 1995).

Menurut Wozniak (1995), telah banyak penelitian yang menunjukkan bahwa variasi waktu pada perulangan berjangka berpengaruh pada kekuatan memori dan hasil penelitian dapat diaplikasikan para pembelajaran yang efektif. Dengan mengubah waktu interval dalam perulangan berjangka, efektifitas pembelajaran dapat diketahui. Kelemahan yang sering ditemukan pada penelitian tentang perulangan berjangka adalah kurangnya perhatian terhadap tingkat

kesulitan *item* tertentu, sebagai contoh, jadwal perulangan yang sama digunakan terhadap *item* dengan tingkat kesulitan "sulit" dan tingkat kesulitan "mudah".

2.5.1.3 Algoritma SM

Tujuan eksperimen selanjutnya adalah membuktikan keberadaan nilai yang paling optimal untuk jarak waktu pada perulangan berjangka atau *optimum inter-repetition intervals* dan menghitung nilainya. Pada akhir eksperimen telah tercipta algoritma SM-0 dengan detil sebagai berikut (Wozniak, 1990)

1. Pilah informasi menjadi *item* terkecil yang mungkin.
2. Asosiasikan *item-item* menjadi grup yang terdiri dari 20-40 elemen.
3. Ulangi masing-masing grup menggunakan interval berikut

   I (1) = 1

   I (2) = 7

   I (3) = 16

   I (4) = 35

   Untuk i > 4 : I(i) = I(i-1)*2

   Dimana I(i) adalah interval waktu yang digunakan setelah perulangan ke-i.
4. Semua item yang terlupakan disalin setelah interval 35 hari ke grup baru (tanpa menghapusnya dari grup yang lama). Grup baru ini akan diulang dengan cara yang sama dengan item yang dipelajari pertama kali.

Sebagai catatan, *inter-repetition intervals* atau jangka waktu antar perulangan untuk perulangan kelima dan seterusnya diasumsikan dua kali lebih lama daripada perulangan sebelumnya. Fakta ini lebih didasari oleh intuisi dibandingkan hasil eksperimen. Setelah 2 tahun menggunakan algoritma SM-0,

data yang cukup telah dikumpulkan untuk memastikan bahwa asumsi tersebut akurat dan beralasan (Wozniak, 1990).

Inter-repetition intervals untuk perulangan setelah I(4) cenderung bernilai konstan (misalnya pada algoritma SM-0 untuk menghafal kosakata dalam bahasa Inggris bernilai sama dengan 2) sehingga terciptalah formula baru untuk menghitung nilai *inter-repetition intervals* sebagai berikut (Wozniak, 1990)

I(1):=1

I(2):=6

Untuk n>2 I(n):=I(n-1)*EF

Di mana :

I(n) adalah *inter-repetition intervals* setelah perulangan ke-n (dalam satuan hari), dan EF adalah *easiness factor* yang menunjukkan tingkat kesulitan *item* tertentu untuk diingat (untuk selanjutnya disebut *E-factor*). Nilai *E-Factor* berkisar antara 1,1 untuk *item* yang paling sulit dan bernilai 2,5 untuk *item* yang paling mudah diingat. Nilai *E-Factor* selalu 2,5 untuk setiap *item* yang baru dimasukkan. Ketika proses mengingat *item* tertentu semakin sulit, maka nilai *E-Factor* semakin menurun. Semakin sulit sebuah *item* untuk diingat, maka penurunan nilai *E-Factor* semakin signifikan (Wozniak, 1990).

Beberapa saat setelah program SuperMemo diimplementasikan, disadari bahwa nilai *E-Factor* tidak diperkenankan berada di bawah 1,3. Item yang memiliki nilai *E-Factor* di bawah 1,3 akan diulang lebih sering dari yang seharusnya. Nilai *E-Factor* yang tidak di bawah 1,3 akan membuat keluaran proses menjadi lebih baik dan menunjukkan indikator bahwa sebuah *item* harus

direformulasi (Wozniak, 1990).

Menurut Wozniak (1990), untuk dapat menghitung nilai baru dari *E-Factor* pengguna harus memberi nilai pada respon dari pertanyaan yang ada pada setiap item selama perulangan (SuperMemo menggunakan skala tingkatan 0 sampai 5). Bentuk umum dari formula yang digunakan adalah

EF' := f(EF, q)

Di mana:

EF' : nilai *E-Factor* yang baru

EF : nilai *E-Factor* yang lama

q : kualitas response

f : fungsi yang digunakan untuk menghitung EF'

Secara singkat, fungsi f memiliki bentuk akhir sebagai berikut (Wozniak, 1990)

EF' := EF - 0,8 + 0,28*q – 0,02*q*q

yang merupakan bentuk sederhana dari

EF' := EF + (0.1 - (5-q)*(0.08 + (5-q)*0.02))

Dari rumus di atas, dapat diketahui bahwa untuk nilai q = 4, maka nilai *E-Factor* tidak berubah.

2.5.2 Mnemosyne

Mnemosyne merupakan aplikasi *flashcard* lain yang menggunakan algoritma SM-2 dengan sedikit modifikasi. Aplikasi ini dibuat oleh Peter Bienstman, seseorang yang dulunya pernah bergabung dalam pembuatan aplikasi *flashcard* MemAid. Aplikasi MemAid sendiri bermula dari SuperMemo, dan kini

menjadi aplikasi yang bersifat komersial dengan nama FullRecall. Walaupun berjenis aplikasi *flashcard*, Mnemosyne yang dikembangkan oleh Peter Bientsman memiliki karakter yang sedikit berbeda dari SuperMemo. Algoritma yang dipakai pada aplikasi *Mnemosyne* 1.0 adalah aplikasi SM-2 atau *SM-algorithm* versi 2 yang telah dimodifikasi. Selain itu, pada Mnemosyne dimungkinkan pengiriman log atau data mengenai *card* yang dipelajari oleh pengguna. Hal ini memungkinkan penelitian lebih lanjut mengenai penentuan *optimum inter-repetition intervals* yang lebih baik.

Masing-masing *card* dalam Mnemosyne memiliki 13 buah atribut (Bientsman, 2008). Atribut-atribut tersebut adalah sebagai berikut

1. ID

   Identifikasi unik dari tiap *card*. Dalam Mnemosyne, setiap *card* memiliki ID yang berupa *hash* dari data.

2. *Grade*

   Grade adalah rentang integer dari 0 hingga 5. Grade 0 menandakan bahwa *card* belum dipelajari atau menggambarkan bahwa *card* sulit diingat. Pemberian *grade* 0 atau 1 pada *card* menandakan pengguna tidak mengetahui jawaban atau *answer* dari *card*, atau dengan kata lain telah melupakan *card*. Perbedaan *grade* 0 dan 1 hanya pada hal rentang waktu kapan *card* akan ditanyakan kembali. *Card* dengan *grade* 0 atau 1 akan terus ditanyakan hingga pengguna memberi *grade* 2 atau lebih pada *card*.

3. *Easiness*

   Easiness menyatakan tingkat kesulitan dari *card*. Rentang nilainya

dipengaruhi oleh *grade* baru yang diberikan oleh pengguna.

4. *Acquisition reps*

   Istilah *acquisition phase* atau fase akuisisi digunakan untuk menggambarkan semua *card* yang memiliki *grade* 0 atau 1. *Acquisition reps* pada *card* menunjukkan seberapa sering proses pembelajaran *card* yang bersangkutan dengan hanya berkutat pada *grade* 0 atau 1.

5. *Retention reps*

   Istilah *retention phase* atau fase retensi digunakan untuk menggambarkan semua *card* yang memiliki *grade* 2 sampai 5. Ini berarti bahwa pengguna dapat dengan baik mengingat *card*. *Retention reps* pada *card* menunjukkan seberapa sering proses pembelajaran *card* yang bersangkutan dengan diberi *grade* 2 hingga 5 oleh pengguna.

6. *Lapses*

   Apabila sebuah *card* telah memasuki fase retensi, ada kemungkinan pengguna melupakan atau tidak dapat dengan baik mengingat *card.* Dengan demikian, *grade* dari *card* yang bersangkutan dapat berubah dari fase retensi ke fase akuisisi. Hal inilah yang disebut dengan istilah *lapse*. Nilai atribut *lapses* pada *card* akan meningkat bila hal ini terjadi.

7. *Acquisition reps since lapse*

   Acquisition reps since lapse adalah rentang waktu *acquisition* sejak *lapse.*

8. *Retention reps since lapse*

   Retention reps since lapse menyatakan rentang waktu *retention* sejak *lapse.*

9. *Last repetition*

Last repetition menggambarkan rentang waktu (dihitung sejak hari dimulainya proses belajar) *card* terakhir kali dimunculkan.

10. *Next repetition*

    Next repetition menggambarkan rentang waktu *card* akan dimunculkan kembali.

11. *Question*

    Berisi pertanyaan dari *card*.

12. *Answer*

    Berisi jawaban dari *card*.

13. Kategori

    Kategori *card* yang ditentukan sendiri oleh pengguna, misalnya: “katakana”, “hiragana”, atau “negara dan ibukotanya”.

2.6 NetBeans IDE

Dalam merancang dan membangun sebuah aplikasi Java, pada umumnya dapat dilakukan dengan menggunakan editor teks seperti Notepad, Vim, Kate, atau Gedit. Pada kenyataannya, jarang sekali ada *programmer* yang menggunakan editor teks biasa. Ini disebabkan fitur-fitur pada editor teks yang terbatas, selain untuk meningkatkan produktivitas kerja dalam proses pengembangan.

Keberadaan *IDE* atau *Integrated Development Environment* menjadi alat bantu yang berharga karena di dalamnya terkadang bukan hanya terdapat fitur pengolah teks, namun juga tampilan serta kemudahan yang tidak didapat di editor teks biasa seperti *auto-completion*, *compiler*, dan sebagainya. *IDE* ada yang bersifat komersial atau *proprietary*, dan ada pula yang bersifat kode sumber

terbuka. Salah satu *IDE* yang populer dalam pengembangan aplikasi Java adalah *NetBeans* dari *Sun.* Saat penulisan penelitian ini dibuat, *NetBeans* telah mencapai versi 6.1.

Fitur dalam NetBeans dapat ditambah dengan menggunakan pengaya. Pengaya untuk membuat *mobile application* paling umum adalah *NetBeans Mobility Pack* yang memungkinkan pemrogram membuat aplikasi JME dengan relatif mudah. Pengaya ini juga sudah mendukung MIDP 2.0 dan CLDC 1.1. Di dalam pengaya ini juga telah dimasukkan emulator, sehingga aplikasi yang dibuat dapat dicoba dulu di komputer sebelum ditanam dalam telepon seluler sebenarnya. Selain itu, sebagai pelengkap digunakan pula *UML Designer*.

2.7 *Unified Modeling Language (*UML*)*

2.7.1 Pengertian UML

Unified Modeling Language (UML) merupakan bahasa standar yang digunakan untuk menulis perencanaan sebuah perangkat lunak. *UML* digunakan untuk memvisualisasikan, membuat spesifikasi, membangun dan mendokumentasikan sistem dari perangkat lunak (Booch, Rumbaugh, & Jacobson, 1998).

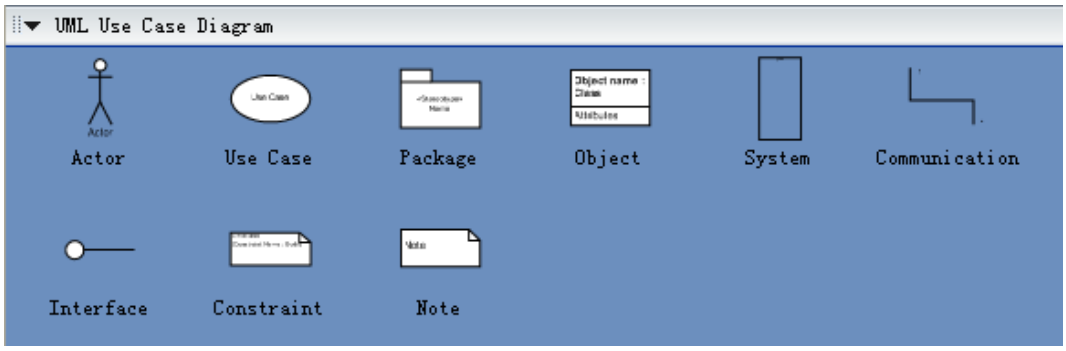
UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi perangkat lunak, di mana aplikasi tersebut dapat berjalan pada perangkat keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan perangkat lunak dalam bahasa

yang berorientasi objek seperti C++, Java, C#, atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C (Dharwiyanti & Wahono, 2003).

2.7.2 *Use case diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya (Dharwiyanti & Wahono, 2003).

Simbol atau notasi yang digunakan dalam *use case diagram* terlihat pada gambar 2.7.



Gambar 2.7. Simbol pada use case diagram (http://edrawsoft.com, 2008)

2.7.3 *Class diagram*

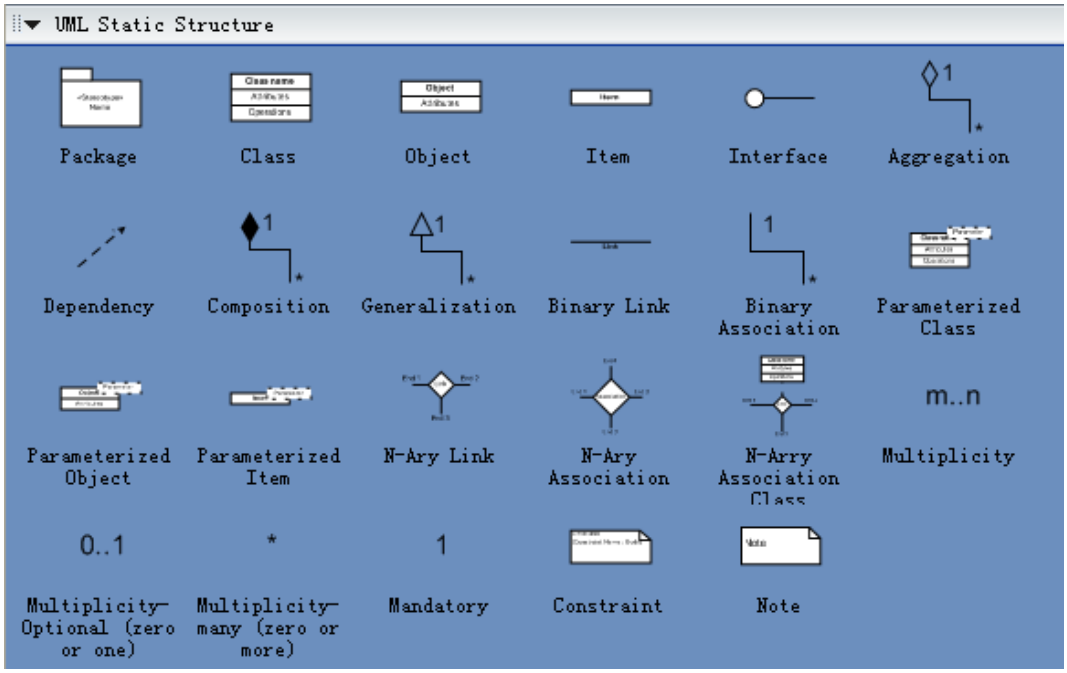
Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus

menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi) (Dharwiyanti & Wahono, 2003).

Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok (Dharwiyanti & Wahono, 2003) :

1. Nama (dan *stereotype*).
2. Atribut
3. Metode

Simbol yang sering digunakan pada *class diagram* terlihat pada gambar 2.8.



Gambar 2.8. Simbol pada *class diagram* (http://edrawsoft.com, 2008)

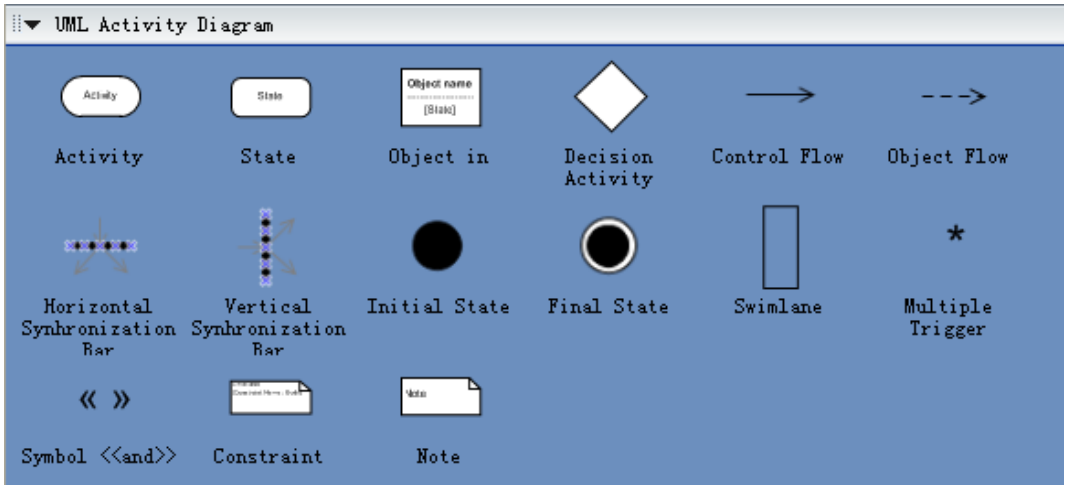
2.7.4 *Activity diagram*

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang

mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi (Dharwiyanti & Wahono, 2003).

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum (Dharwiyanti & Wahono, 2003).

Simbol yang digunakan pada *activity diagram* ditunjukkan oleh gambar 2.9.



Gambar 2.9. Simbol yang digunakan pada *activity diagram* (http://edrawsoft.com, 2008)