II. TINJAUAN PUSTAKA

A. Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*object-oriented programming*) merupakan paradigma pemrograman yang berorientasikan kepada objek. Fokus utama diberikan kepada objek-objek yang terlibat dalam penyelesaian dan juga bagaimana mereka bekerja sama dalam menyelesaikan suatu masalah. Dengan menggunakan OOP dalam melakukan pemecahan suatu masalah, kita tidak hanya melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Semua data dan fungsi di dalam paradigma ini dikemas dalam kelas-kelas atau objek-objek. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya.

Pemrograman berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, para pengguna OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Bahasa pemrograman *Java* merupakan salah satu bahasa pemrograman yang berorientasi objek Untuk dapat menguasai pemrograman *Java*, harus mengerti dengan baik konsep

pemrograman berorientasi objek. Berikut ini diuraikan konsep-konsep penting dalam pemrograman berorientasi objek, sehingga diharapkan kita akan lebih mudah dalam mempelajari bahasa pemrograman *Java* (Sanchez, Julio dan Canton, Maria P, 2002).

1. Objek

Pada dasarnya semua benda yang ada di dunia nyata dapat dianggap sebagai sebuah objek. Terdapat dua karakteristik yang utama pada sebuah objek, yaitu :

a) Setiap objek memiliki atribut sebagai status yang kemudian akan disebut sebagai *state.*
b) Setiap objek memiliki tingkah laku yang kemudian akan disebut sebagai *behaviour*.

Dalam pengembangan perangkat lunak berorientasi objek, objek dalam perangkat lunak akan menyimpan *state*-nya dalam variabel dan menyimpan informasi tingkah laku (*behaviour*) dalam *method-method* atau fungsi-fungsi.

2. Kelas (*Class*)

Kelas merupakan prototipe yang mendefinisikan variabel-variabel dan *method-method* secara umum. Kelas juga bisa berarti kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh "*class of cat*“ adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku atau turunan dari kucing. Sebuah kelas adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Dengan modularitas,

struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

3. Abstraksi

Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari pelaku abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

4. Enkapsulasi

Dalam sebuah objek yang mengandung variabel-variabel dan *method-method*, dapat ditentukan hak akses pada sebuah variabel atau *method* dari objek. Pembungkusan variabel dan *method* dalam sebuah objek dalam bagian yang terlindungi inilah yang disebut dengan enkapsulasi. Jadi, enkapsulasi dapat diartikan sebagai bungkusan (*wrapper*) pelindung program dan data yang sedang diolah. Pembungkus ini mendefinisikan perilaku dan melindungi program dan data yang sedang diolah agar tidak diakses sembarangan oleh program lain. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

Manfaat dari proses enkapsulasi adalah :

a) Modularitas, adalah kode sumber dari sebuah objek dapat dikelola secara independen dari kode sumber objek yang lain.
b) *Information Hiding*, adalah kita dapat menentukan hak akses sebuah variabel atau *method* dari objek, dengan demikian kita bisa menyembunyikan informasi yang tidak perlu diketahui objek lain.

5. *Inheritance* (Pewarisan)

Kelas dapat didefinisikan dengan referensi pada kelas yang lain yang telah terdefinisi. *Inheritance* merupakan pewarisan atribut dan *method* pada sebuah kelas yang diperoleh dari kelas yang telah terdefinisi tersebut. Setiap subkelas akan mewarisi *state* (variabel-variabel) dan *behaviour* (*method-method*) dari superkelas-nya. Subkelas kemudian dapat menambahkan *state* dan *behaviour* baru yang spesifik dan dapat pula memodifikasi (*override*) *state* dan *behaviour* yang diturunkan oleh superkelas-nya.

Keuntungan dari *inheritance* adalah :

a) Subkelas menyediakan *state* atau *behaviour* yang spesifik yang membedakannya dengan superkelas, hal ini akan memungkinkan pemrogram *Java* untuk menggunakan ulang *source code* dari superkelas yang telah ada.
b) Pemrogram *Java* dapat mendefinisikan superkelas khusus yang bersifat generik, yang disebut *abstract class*, untuk mendefinisikan kelas dengan *behaviour* dan *state* secara umum.

Istilah dalam *inheritance* yang perlu diperhatikan :

1. *Extends*, kata kunci ini harus kita tambahkan pada definisi kelas yang menjadi subkelas.
2. *Superclass* (superkelas), superkelas digunakan untuk menunjukkan hirarki kelas yang berarti kelas dasar dari subkelas.
3. *Subclass* (subkelas), subkelas adalah kelas anak atau turunan secara hirarki dari superkelas.
4. Super, kata kunci ini digunakan untuk memanggil konstruktor dari superkelas atau menjadi variabel yang mengacu pada superkelas.
5. *Method Overriding*, pendefinisian ulang *method* yang sama pada subkelas.

Dalam *inheritance*, *method overriding* berbeda dengan *method overloading*. *Method overriding* adalah mendefinisikan kembali *method* yang sama, baik nama *method* maupun *signature* atau parameter yang diperlukan dalam subkelas, Sedangkan *method overloading* adalah mendefinisikan *method* yang memiliki nama yang sama, tetapi dengan *signature* yang berbeda dalam definisi kelas yang sama.

6. Polimorfisme

Kata polimorfisme yang berarti satu objek dengan banyak bentuk yang berbeda, adalah konsep sederhana dalam bahasa pemrograman berorientasi objek yang berarti kemampuan dari suatu variabel referensi objek untuk memiliki aksi berbeda bila *method* yang sama dipanggil, di mana aksi *method* tergantung pada tipe objeknya. Kondisi yang harus dipenuhi supaya polimorfisme dapat diimplementasikan adalah :

1. *Method* yang dipanggil harus melalui variabel dari basis kelas atau superkelas
2. *Method* yang dipanggil harus juga menjadi *method* dari basis kelas.
3. *Signature method* harus sama baik pada superkelas maupun subkelas.
4. *Method access attribute* pada subkelas tidak boleh lebih terbatas dari basis kelas.

7. *Interface*

Pada *Java* juga dikenal konsep *interface*, yang merupakan *device* yang digunakan untuk komunikasi antar objek berbeda yang tidak memiliki hubungan apapun. *Interface* bisa dikatakan sebagai protokol komunikasi antar objek tersebut.

B. Teknologi Java

Bahasa pemrograman *Java* dapat dikategorikan sebagai sebuah bahasa pemrograman berorientasi objek, pemrograman terdistribusi dan bahasa pemrograman *multithreaded*. Selain itu *Java* juga bisa diartikan sebagai suatu bahasa pemrograman berorientasi objek dengan unsur-unsur seperti bahasa C++ dan bahasa-bahasa lainnya dengan *libraries* yang cocok untuk lingkungan internet. Objek pada *Java* dispesifikasi dengan membentuk kelas. Sebagai sebuah bahasa pemrograman, *Java* dapat membuat seluruh bentuk aplikasi, *desktop*, *web* dan lainnya, sebagaimana dibuat dengan menggunakan bahasa pemrograman konvensional yang lain.

Java adalah bahasa pemrograman yang berorientasi objek (OOP) dan dapat dijalankan pada berbagai platform sistem operasi. Perkembangan *Java* tidak hanya terfokus pada

satu sistem operasi, tetapi dikembangkan untuk berbagai sistem operasi dan bersifat *open source*. Untuk masing-masing kelas *Java*, *compiler Java* memproduksi sebuah file keluaran arsitektur netral yang akan jalan pada berbagai implementasi dari *Java Virtual Machine* (JVM).

Java menurut definisi dari Sun adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *stand alone* ataupun pada lingkungan jaringan. Java pertama lahir dari The Green Project, yang berjalan selama 18 bulan, dari awal tahun 1991 hingga musim panas 1992. Proyek tersebut belum menggunakan versi yang dinamakan Oak. Proyek ini dimotori oleh Patrick Naughton, Mike Sheridan, James Gosling dan Bill Joy, beserta sembilan pemrogram lainnya dari Sun Microsystems. Salah satu hasil proyek ini adalah maskot *Duke* yang dibuat oleh Joe Palrang.

Pertemuan proyek berlangsung di sebuah gedung perkantoran *Sand Hill Road* di Menlo Park. Sekitar musim panas 1992 proyek ini ditutup dengan menghasilkan sebuah program *Java Oak* pertama, yang ditujukan sebagai pengendali sebuah peralatan dengan teknologi layar sentuh (*touch screen*), seperti pada PDA sekarang ini. Teknologi baru ini dinamai "*7" (*Star Seven*).

Salah satu teknologi yang ditawarkan oleh Java adalah *Write Once Run anywhere* , Java 2 Micro Edition (J2ME) adalah lingkungan pengembangan yang didesain untuk meletakkan perangkat lunak Java pada barang elektronik beserta perangkat pendukungnya. Dengan ada nya program J2ME kita dapat membuat program yang dapat diinstalasi pada Palm, Nokia 9210, Motorola i85s dan *Java Handheld* lainnya.

1. Karakteristik Java

Java mempunyai beberapa karakteristik yang menjadikannya berbeda dengan bahasa pemrograman lain. Pada tabel berikut ini dijelaskan karakteristik bahasa pemrograman *Java.*

Tabel 1. Karakteristik Bahasa Pemrograman Java

Karakteristik	Penjelasan
Simple	Bahasa pemrograman *Java* menggunakan sintaks mirip dengan C++ namun sintaks pada *Java* telah banyak diperbaiki terutama menghilangkan penggunaan *pointer* yang rumit dan *multiple inheritance*. *Java* juga menggunakan *automatic memory allocation* dan *memory garbage collection*.
Object Oriented	*Java* menggunakan pemrograman berorientasi objek yang membuat program dapat dibuat secara modular dan dapat dipergunakan kembali. Pemrograman berorientasi objek memodelkan dunia nyata ke dalam objek dan melakukan interaksi antar objek-objek tersebut.
Distributed	*Java* dibuat untuk membuat aplikasi terdistribusi secara mudah dengan adanya *libraries networking* yang terintegrasi pada *Java*

Tabel 1. Lanjutan

Interpreted	Program *Java* dijalankan menggunakan interpreter yaitu *Java Virtual Machine* (JVM). Hal ini menyebabkan *source code Java* yang telah dikompilasi menjadi *Java bytecodes* dapat dijalankan pada *platform* yang berbeda.
Robust	*Java* mempuyai reliabilitas yang tinggi. *Compiler* pada *Java* mempunyai kemampuan mendeteksi kesalahan secara lebih teliti dibandingkan bahasa pemrograman lain. *Java* mempunyai *runtime-Exception handling* untuk membantu mengatasi kesalahan pada pemrograman.
Secure	Sebagai bahasa pemrograman untuk aplikasi internet dan terdistribusi, *Java* memiliki beberapa mekanisme keamanan untuk menjaga aplikasi tidak digunakan untuk merusak sistem komputer yang menjalankan aplikasi tersebut.
Architecture Neutral	Program *Java* merupakan *platform independent*. Artinya program cukup mempunyai satu buah versi yang dapat dijalankan pada *platform* berbeda dengan *Java Virtual Machine*.
Portable	*Source code* maupun program *Java* dapat dengan mudah dibawa ke *platform* yang berbeda-beda tanpa harus dikompilasi ulang.
Performance	*Performance* pada *J*ava sering dikatakan kurang tinggi. Namun *performance Java* dapat ditingkatkan menggunakan kompilasi *Java* lain seperti buatan Inprise, Microsoft ataupun Symantec yang menggunakan *Just In Time Compilers* (JIT).
Multithreaded	*Java* mempunyai kemampuan untuk membuat suatu program yang dapat melakukan beberapa pekerjaan secara sekaligus dan simultan.
Dynamic	*Java* didesain untuk dapat dijalankan pada lingkungan yang dinamis. Perubahan pada suatu kelas dengan menambahkan *properties* ataupun *method* dapat dilakukan tanpa menggangu program yang menggunakan kelas tersebut.

2. Beberapa Fitur yang Terdapat pada *Java*

Berikut ini merupakan beberapa fitur yang terdapat pada bahasa pemrograman *Java* beserta penjelasannya.

a. *Java Virtual Machine* (JVM)

JVM adalah sebuah mesin imajiner (maya) yang bekerja dengan menyerupai aplikasi pada sebuah mesin nyata. JVM menyediakan spesifikasi perangkat keras dan *platform* di mana kompilasi kode *Java* terjadi. Spesifikasi inilah yang membuat aplikasi berbasis *Java* menjadi bebas dari *platform* manapun karena proses kompilasi diselesaikan oleh JVM.

Aplikasi program *Java* diciptakan dengan file teks berekstensi. `.java`. Program ini dikompilasi menghasilkan satu berkas *bytecode* berekstensi `.class` atau lebih. *Bytecode* adalah serangkaian instruksi serupa instruksi kode mesin. Perbedaannya adalah kode mesin harus dijalankan pada sistem komputer di mana kompilasi ditujukan, sementara *bytecode* berjalan pada *Java interpreter* yang tersedia di semua *platform* sistem komputer dan sistem operasi.

b. *Garbage Collection*

Banyak bahasa pemrogaman lain yang mengizinkan seorang pemrogram mengalokasikan memori pada saat dijalankan. Namun, setelah menggunakan alokasi memori tersebut, harus terdapat cara untuk menempatkan kembali blok memori tersebut agar program lain dapat menggunakannya. Dalam C, C++, dan bahasa lainnya, adalah pemrogram yang

mutlak bertanggung jawab akan hal ini. Hal ini dapat menyulitkan bilamana pemrogram tersebut lupa untuk mengembalikan blok memori sehingga menyebabkan situasi yang dikenal dengan nama *memory leaks*.

Program *Java* melakukan *garbage collection* yang berarti program tidak perlu menghapus sendiri objek – objek yang tidak digunakan lagi. Fasilitas ini mengurangi beban pengelolaan memori oleh pemrogram dan mengurangi atau mengeliminasi sumber kesalahan terbesar yang terdapat pada bahasa yang memungkinkan alokasi dinamis.

c. *Code Security*

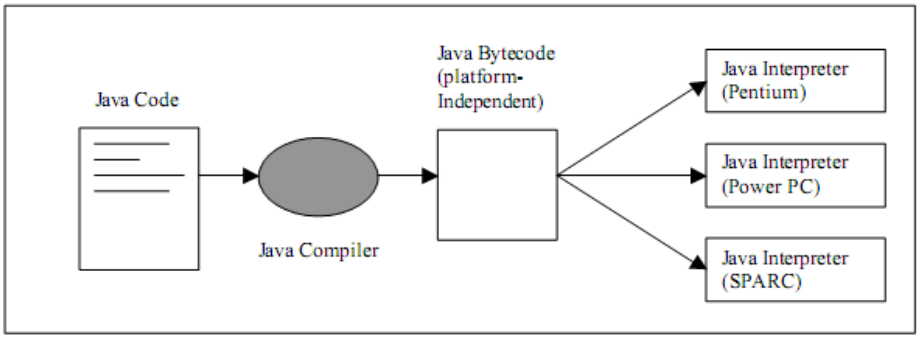
Code Security terimplementasi pada *Java* melalui penggunaan *Java Runtime Environment* (JRE). *Java* menggunakan model pengamanan tiga lapis untuk melindungi sistem dari *Untrusted Java Code*.

1. *Class-loader* menangani pemuatan kelas *Java* ke *runtime interpreter*. Proses ini menyediakan pengamanan dengan memisahkan kelas – kelas yang berasal dari *local disk* dengan kelas – kelas yang diambil dari jaringan. Hal ini membatasi aplikasi trojan atau virus karena kelas – kelas yang berasal dari *local disk* yang dimuat terlebih dahulu.
2. *Bytecode verifier* membaca *bytecode* sebelum dijalankan dan menjamin *bytecode* memenuhi aturan – aturan dasar bahasa *Java*.
3. Manajemen keamanan menangani keamanan tingkat aplikasi dengan mengendalikan apakah program berhak mengakses sumber daya seperti sistem file, *port* jaringan, proses eksternal, dan sistem *windowing*.

Setelah seluruh proses tersebut selesai dijalankan, barulah kode program dieksekusi.

3. Cara Kerja *Java*

Pada pemrograman menggunakan *Java* kita menggunakan *compiler* sekaligus *interpreter* agar dapat program kita dapat berjalan pada *platform* yang berbeda. *Java compiler* melakukan kompilasi pada *source code* menjadi *Java bytecodes*. *Java bytecodes* yang merupakan instruksi mesin yang tidak spesifik terhadap processor pada sistem komputer akan dijalankan pada *platform* menggunakan *Java Virtual Machine* (JVM) yang disebut juga *bytecodes interpreter* atau *Java runtime interpreter*. Ini bisa dilihat pada gambar.



Gambar 1. Cara Kerja *Java*

Dari gambar dapat dijelaskan bahwa langkah pertama dalam pembuatan sebuah program berbasis *Java* adalah menuliskan kode program pada *text editor* (*notepad,* vi, *emacs* dan lain sebagainya). Kode program yang dibuat kemudian tersimpan dalam sebuah file berekstensi `.java`. Setelah membuat dan menyimpan kode program, kompilasi file yang berisi kode program tersebut dengan menggunakan *Java Compiler*. Hasil dari kompilasi adalah berupa file *bytecode* dengan ekstensi `.class`. File yang mengandung *bytecode*

tersebut kemudian akan dikonversikan oleh *Java Interpreter* menjadi bahasa mesin sesuai dengan jenis dan *platform* yang digunakan.

Tabel 2. Cara Kerja Java

Proses	Tool	Hasil
Menulis kode program	Teks *editor*	File berekstensi `.java`
Kompilasi program	*Java Compiler*	File berekstensi `.class` (*Java Bytecodes*)
Menjalankan program	*Java Interpreter*	Keluaran Program

C. Perkembangan Java

Berdasarkan referensi yang diperoleh dari penulis buku *Pemrograman Aplikasi Wireless dengan Java*, Ady Wicaksono, bahasa Java awalnya adalah bernama Oak, yakni bagian dari projek Green yang dikembangkan khusus oleh *Sun Microsystem* untuk memprogram perangkat elektronik rumah tangga semacam televisi. Namun pada perkembangannya, bahasa Oak ini menjadi bahasa yang bisa digunakan untuk pemrograman secara umum dan dikenal menjadi bahasa Java saat ini.

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 Environtment*) yang dikhususkan untuk menjalankan program program Java dan JSDK (*Java Software Develpoment Kit*) atau JDK yang terdiri atas paket paket yang bisa digunakan untuk mengkompilasi program program dengan bahasa Java sekaligus menjalankannya.

Pada awal rilisnya, JDK 1.0.2 membuat gebrakan dalam dunia *web* sekalipun saat ini, teknologi yang dimaksud sudah relatif jarang digunakan oleh orang, yakni Java Applet. Pada versi 1.1, beberapa fitur baru semacam Java Swing, Java RMI (*Remote Method Invocation)* dan JIT (*Just In Time) compiler* dikenalkan. Selain itu beberapa perbaikan pada fitur sebelumnya juga dilakukan. Pada perkembangan selanjutnya, *Sun Microsystem* memperkenalkan Java versi 1.2 atau lebih kenal dengan *Java 2 Compliant.* Pada Java 2 ini, Java ini dibagi menjadi tiga kategori, yaitu :

1. Java 2 Standard Edition (J2SE)

Kategori ini digunakan untuk menjalankan dan mengembangkan aplikasi Java pada level PC.

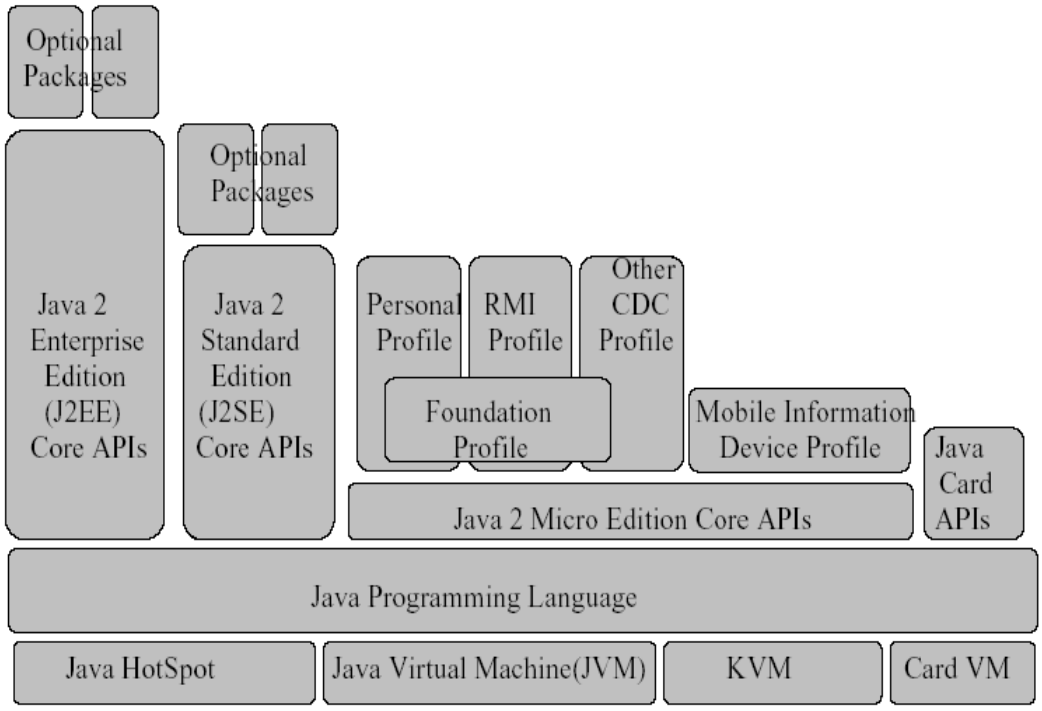
2. Java 2 Enterprise Edition (J2EE)

Digunakan untuk menjalankan dan mengembangkan aplikasi Java pada lingkungan *enterprise,* dengan menambah fungsionalitas fungsionalitas Java semacam EJB (*Enterprise Java Bean),* Java CORBA, *Servlet* dan JSP, serta Java XML (*Extensible Markup Language).*

3. Java 2 Micro Edition (J2ME)

Kategori ini digunakan untuk menjalankan dan mengembangkan aplikasi Java pada *handheld devices* atau perangkat perangkat semacam handphone, Palm, PDA dan Pocket PC. Namun sebenarnya saat ini ada kategori lain yakni Java Card, yang khusus dikembangkan untuk membuat aplikasi aplikasi pada *smart Card*, misalnya aplikasi kartu

telephone CHIP, kartu VISA, kartu SIM pada *handphone* dan aplikasi *mobile banking* BCA yang saat ini sudah umum digunakan



Gambar 2. Lingkungan Kerja Teknologi Java

D. Pengenalan J2ME

Java 2 Micro edition (J2ME) merupakan *subset* dari J2SE yang ditujukan untuk implementasi pada peralatan *embedded system* dan *handheld* yang tidak mampu mendukung secara penuh implementasi menggunakan J2SE. J2ME merupakan sebuah kombinasi yang terbentuk antara sekumpulan *interface Java* yang sering disebut dengan *Java* API (*Aplication Programming Interface*) dengan JVM (*Java Virtual Machine*) yang didesain khusus untuk alat, yaitu JVM dengan ruang yang terbatas. Kombinasi tersebut

kemudian digunakan untuk melakukan pembuatan aplikasi-aplikasi yang dapat berjalan di atas alat. Kita tidak harus melakukan instalasi JVM dan *Java* API ke dalam alat yang akan kita gunakan karena masing-masing dari perusahaan alat telah menyediakan JVM dan sekumpulan *Java* API yang diperlukan di dalam alat bersangkutan. Hal ini membuat kita sebagai pengembang hanya perlu berkonsentrasi dalam pengembangan aplikasinya dan memasukkannya ke dalam alat tersebut.

Embedded system adalah produk-produk dengan komputer kecil berada di dalamnya, namun aplikasi yang dapat dimanfaatkan dari peralatan tersebut sangatlah spesifik. Hal ini tentu saja berbeda dengan komputer PC yang kita kenal sehari-hari, yang mampu digunakan untuk berbagai aplikasi. Contoh *embedded system* yang ada misalnya adalah aplikasi-aplikasi yang memanfaatkan mikroprosesor seperti televisi, sistem keamanan gedung, dan sebagainya. J2ME sangat berguna untuk membangun sebuah aplikasi pada peralatan dengan jumlah memori dan kapasitas penyimpanan yang terbatas, serta kemampuan *user interface* yang terbatas seperti pada perangkat komunikasi bergerak berupa *handphone*, PDA, dan sebagainya. Seperti aplikasi java umumnya yang menggunakan JVM, dalam J2ME digunakan pula *virtual machine* yang disebut K *virtual machine.*

K *virtual machine* adalah *virtual machine* yang sangat kecil dalam kebutuhan memorinya. Huruf K dalam K *virtual machine* adalah singkatan dari kilobyte, untuk menggambarkan betapa *virtual machine* ini bekerja pada total memori yang sedemikian kecil mulai dari 128 kilobyte hingga maksimal rata-rata sekitar 512 kilobyte.

J2ME memiliki beberapa keunggulan yaitu :

1. Sebagaimana kekhasan aplikasi yang ditulis dengan bahasa pemrograman Java, maka aplikasi J2ME memiliki ciri *running any where*, *any time*, *over any device*.
2. Aplikasi dapat dijalankan secara *on-line* maupun *off-line*.
3. Memiliki kode yang *portable*.
4. *Safe network delivery*.
5. Aplikasi yang ditulis dengan J2ME akan memiliki kompatibilitas yang tinggi dengan *platform* J2SE dan J2EE.

E. Jenis Aplikasi Pada J2ME

Pada saat ini terdapat dua jenis aplikasi dari Java 2 Micro Edition (J2ME), yaitu :

A. Walled garden application, yaitu aplikasi yang berdiri sendiri atau *stand-alone* yang berjalan pada *handphone* tanpa perlu mengakses sumber data eksternal melalui jaringan pembawa atau *carrier network*. Contoh dari aplikasi ini adalah kalkulator atau *single player games*.

B. Network aware application atau aplikasi yang berinteraksi dengan jaringan. Tidak seperti aplikasi yang pertama, aplikasi ini memiliki kemampuan untuk mengakses sumber data eksternal. Contoh aplikasi jenis ini adalah aplikasi *e-mail* yang berada di dalam *handphone*, aplikasi untuk mendapatkan kembali data alamat-alamat yang tersimpan melalui jaringan, dan pengiriman *email* berbagai alamat melalui jaringan data.

Seperti telah disebutkan sebelumnya, J2ME dirancang untuk dapat menjalankan program Java pada perangkat semacam *handphone* dan PDA, yang memiliki karakteristik yang berbeda dengan sebuah komputer biasa, misalnya kecilnya jumlah memori pada *handphone* dan PDA. J2ME terdiri atas komponen komponen sebagai berikut :

1. *Java Virtual Machine (JVM),* Komponen ini untuk menjalankan program Java Pada emulator atau *handheld devices*.
2. *Java API (Application Programming Interface),* Komponen ini merupakan kumpulan librari untuk menjalankan dan mengembangkan program Java pada *handheld devices.*
3. *Tools* lain untuk pengembangan aplikasi Java, semacam emulator *Java Phone*, emulator Motorolla.

J2ME dibagi menjadi dua buah bagian yang dikenal dengan istilah *configuration dan profile.* Ada dua kategori J2ME *Configuration* saat ini, yakni CLDC (*Connected Limited Device Configuration*) dan CDC (*Connected Device Configuration*).

Tabel 3. Tabel Perbandingan CLDC dengan CDC

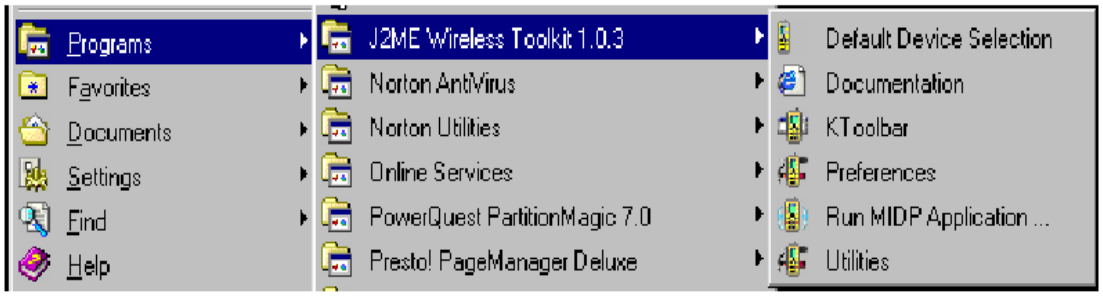
CLDC (Connected Limited Device Configuration)	CDC (Connected Device Configuration)
Mengimplementasikan *subset* dari J2SE	Mengimplementasikan seluruh fitur dari J2SE
JVM yang digunakan adalah KVM	JVM yang digunakan adalah CVM
Digunakan pada perangkat handheld (handphone, PDA, *two way pager*) dengan memory terbatas (160-512 KB)	Digunakan pada perangkat *handheld* (*internet* TV, Nokia *communicator, car* TV) dengan *memory* minimal 2 MB
Prosesor : 16/ 32 bit	Prosesor : 32 bit

Sedangkan J2ME *Profile* menyediakan implementasi tambahan yang sangat spesifik dari sebuah *handheld devices.* Saat ini, terdapat lima kategori dari J2ME *Profile,* yaitu :

1. *Mobile Information Device Profile* (MIDP)
2. *Foundation Profile* (FP)
3. *Personel Profile*
4. RMI *Profile*
5. *Personel Digital Assistance Profile*

F. J2ME Wireless Toolkit

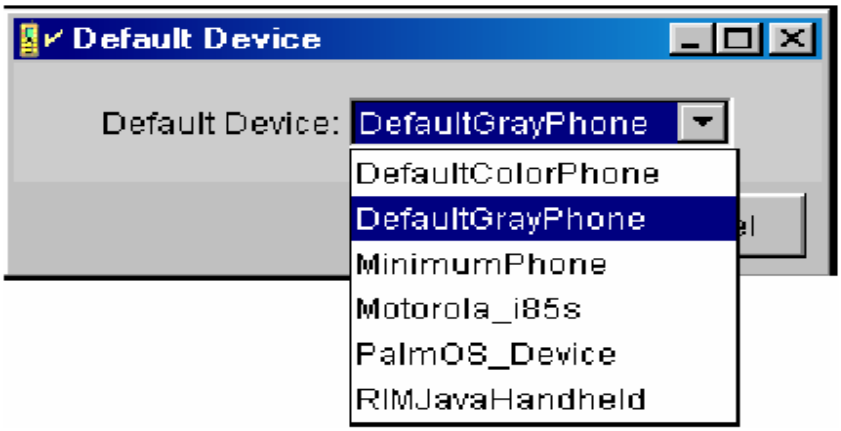
Menu yang terdapat pada J2ME *Wireless Toolkit* terlihat pada gambar di bawah ini :



Gambar 3. Menu J2ME Wireless Toolkit

1. Menu Default Device Selection

Menu *default device selection,* digunakan untuk memilih *default emulator* yang akan digunakan saat mencoba aplikasi *wireless* Java nantinya. Masing masing emulator memiliki karakteristik tersendiri, namun memiliki fitur standar sebagai sebuah *handheld.*

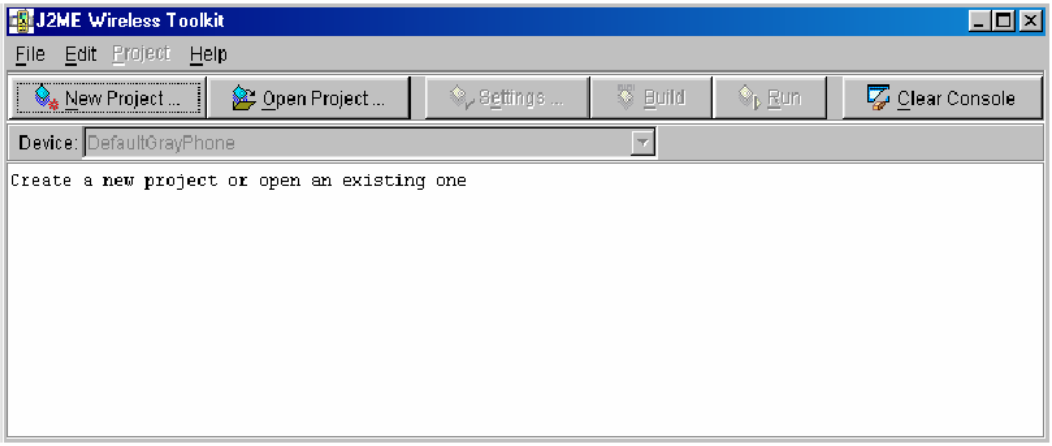


Gambar 4. Menu Default Device Selection

Gambar 5. Contoh Emulator Default GrayPhone

2. Menu Ktoolbar

Menu ini merupakan menu utama dari aplikasi untuk pengembangan Java MIDP atau lebih dikenal dengan istilah MIDlet. Ktoolbar merupakan lingkungan pengembangan minimal yang disediakan oleh J2ME Wireless Toolkit untuk pengembangan MIDlet.



Gambar 6. Ktoolbar

G. Dasar MIDlet

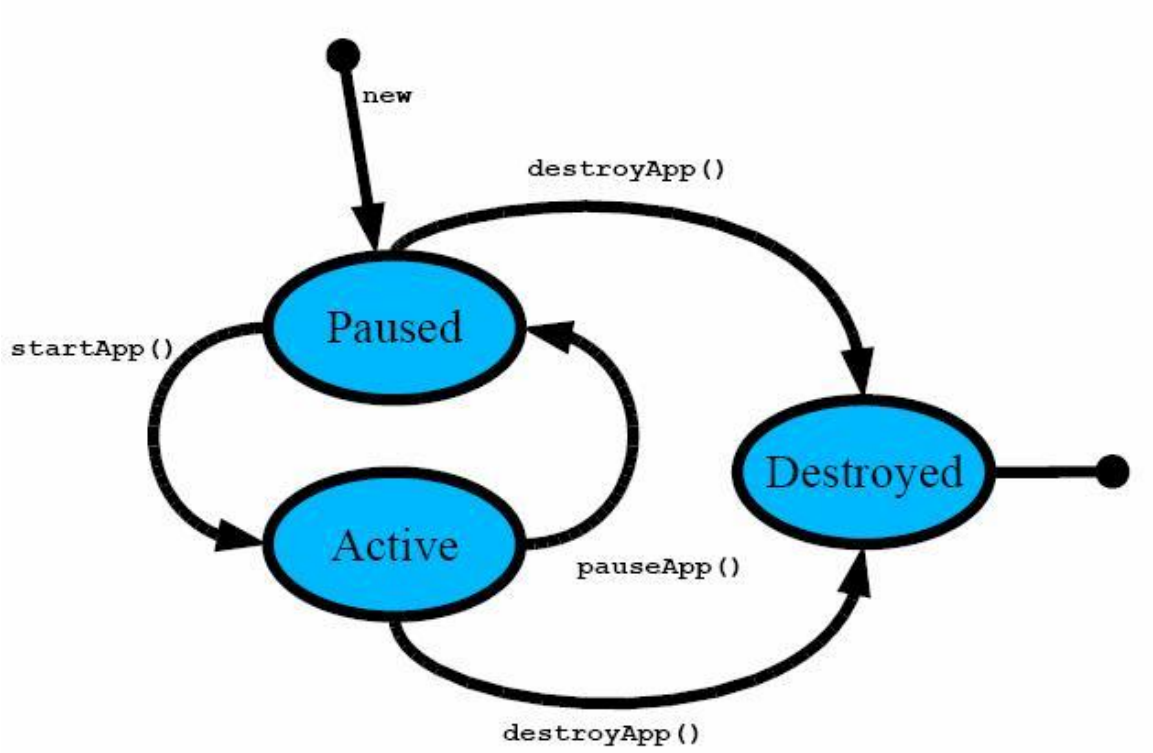
Midlet adalah aplikasi yang dibuat menggunakan J2ME dengan *profile* MIDP. MIDP dikhususkan untuk digunakan pada perangkat dengan kemampuan CPU, memori keyboard, dan layar yang terbatas, misalnya saja *ponsel*.

1. Status MIDlet

MIDlet merupakan istilah yang diberikan untuk aplikasi java J2ME yang menggunakan profile MIDP (*Mobile Information Device Profile*) pada implementasinya. MIDlet

memiliki beberapa status, yaitu *Pause, Active dan Destroy*. Dalam status *Pause*, MIDlet diinisialisasi dan tidak melakukan aksi apapun, pemanggil fungsi pause yaitu *pauseApp().* S*tatus* Active terjadi ketika MIDlet sedang aktif, yakni dipanggil dengan fungsi *startApp().* Sedangkan status *Destroyed* terjadi ketika MIDlet berhenti berjalan, pemanggil fungsinya menggunakan *destroyApp()*

daur hidup MIDlet :



Gambar 7. Daur hidup MIDlet

2. MIDlet Suite dan Application Descriptor

Ketika selesai membuat aplikasi, aplikasi tersebut harus dikemas dalam sebuah paket yang berekstensi *.*jar*. Selain itu dikenal juga file **.jad* yang disebut sebagai *application descriptor* yang berguna untuk mendeskripsikan isi dari file JAR tadi.

H. Model Pemrograman GUI (*Graphical User Interface*) pada MIDlet

Untuk menampilkan sesuatu pada layar ponsel dibutuhkan fungsi *display* dan *screen* yang keduanya terdapat di kelas *javax.microedition.lcdui.*

1. Display

Kelas *display* merupakan kelas yang menyediakan fungsi-fungsi untuk manajemen layar, menampilkan objek *screen* dan menyediakan informasi tentang properti dari perangkat yang digunakan. Akses ke layar dapat diacu dengan fungsi *getDisplay()* pada kelas *display* : public static Display getDisplay().

Penggunaaan fungsi ini umumnya dilakukan di dalam fungsi *startApp()* di mana sebuah MIDlet menjadi aktif. Selain itu ada fungsi *setCurrent*, yang berfungsi untuk menentukan objek *screen* mana yang akan ditampilkan.

2. Screen

Fungsi *screen* menyediakan fungsi interaksi antara pengguna dengan perangkat ponselnya. Objek *screen* mempunyai empat jenis objek yaitu *Textbox, Alert, List* dan *Form.* Objek *screen* bisa memiliki *title* dan *ticker.*

a. List

Kelas *List* menyediakan masukkan pilihan pada layar. Konstruktor dari kelas ini adalah :

public List (String title, int listType);

public List (String title, int listType,

String [] listElement, Image[] listImage;

Tipe dari *List* yaitu *Exclusive* yang hanya terdapat satu buah pilihan. *Implicit* juga hanya terdapat satu pilihan tetapi penampilannya berbeda. *Multiple* berarti bisa memilih lebih dari satu pilihan. Jika digunakan konstruktor yang pertama, maka akan terdapat *list* kosong. Dalam *list* ini dapat ditambahkan, disisipkan dan diganti pilihannya. Fungsi-fungsi penting yang berkaitan dengan *list* adalah :

1. *public int append (String element, Image image*), Menambahkan satu elemen pada daftar pilihan dengan gambar tertentu. Jika tidak akan menampilkan gambar, parameter *image* bisa di *set null*.
2. *public void insert (int index, String element, Image image*), Menyisipkan satu elemen pada daftar pilihan dengan gambar tertentu pada lokasi yang ditentukan oleh nilai parameter index.
3. *public void set (int index, String element, Image image*), Menetapkan satu elemen pada daftar pilihan di lokasi yang ditentukan oleh nilai *parameter* index dengan gambar tertentu.
4. *public Image getImage (int index),* Menampilkan *image* yang digunakan pada lokasi seperti yang disebutkan pada parameter index.
5. *public String getString (int index*), Menghasilkan string teks yang ada pada objek list pada lokasi index.
6. *public boolean isSelected (int index*), Menghasilkan nilai *true* jika elemen ke-index dipilih oleh pengguna, jika tidak maka fungsi akan menghasilkan nilai *false*.
7. *public void delete (int index),* Menghapus elemen ke-*index* dari daftar pilihan.

8. public int getSelectedIndex(), Menghasilkan nilai index lokasi di mana sebuah elemen dipilih.
9. *public int size()*, Menghasilkan nilai jumlah elemen dalam *list*.

b. Form

Bekerja dengan *form* memungkinkan untuk menampilkan beberapa komponen GUI semacam *list, textbox* dalam satu layar. *Form* dapat menampung komponen-komponen yang disebut *item* dalam satu layar.

Konstruktor dari *form* adalah :

public Form (String title) ;

public Form (String title, Item[] items);

Konstruktor pertama menyediakan sebuah form dengan judul *form*, sedangkan konstruktor kedua mendefinisikan butir-butir apa saja yang akan ada di *form* yang akan dibuat. Komponen-komponen turunan dari kelas *item* yaitu:

1. Choice Group

Kelas ini menyediakan komponen untuk daftar pilihan. Fungsi yang ada di sini sama dengan yang ada dalam fungsi *list*.

2. Date Field

Kelas ini berupa masukkan informasi waktu dan tanggal. Tipe masukan yang digunakan dapat berupa waktu, tanggal maupun keduanya dan zona waktu.

3. Gauge

Objek *gauge* merepresentasikan grafik batang yang disusun secara horizontal yang dapat digunakan dalam *form* untuk merepresentasikan jalannya sebuah proses.

4. String Item

String *item* digunakan untuk meletakkan objek teks string yang tidak bisa diubah oleh pengguna secara langsung pada *form*.

5. Text Field

Object *textfield* digunakan untuk meletakkan objek teks string yang bisa diubah oleh pengguna secara langsung pada *form*.

6. Image

Objek *image* digunakan untuk meletakkan objek gambar yang bisa bersifat dapat diubah-ubah ataupun yang bersifat tidak dapat diubah-ubah oleh aplikasi. Konstruktor dari kelas image ini bersifat statik, sehingga tidak digunakan operator *new* untuk membuat objek *image*, melainkan mengacu ke fungsi *createImage()* pada objek *image* ini :

public static createImage (String name) ;

public static createImage (Image image);

public static createImage (int width, int height);

Konstruktor pertama dan kedua digunakan untuk membuat objek yang bersifat tidak dapat diubah-ubah. Sebagai tambahan, bahwa file gambar yang didukung saat ini adalah hanya file dengan format PNG (*Portable Network Graphics*).

7. Image Item

Image item digunakan untuk mengontrol objek *image* yang ada pada form. Konstruktor dari kelas *ImageItem* ini adalah:

Public void ImageItem(String label, Image img, int layout, String altText);

Ada beberapa nilai yang bisa digunakan untuk parameter layout, yaitu:

LAYOUT_CENTER, LAYOUT_DEFAULT,

LAYOUT_LEFT, LAYOUT_RIGHT,

LAYOUT_NEWLINE_AFTER, LAYOUT_NEWLINE_BEFORE.

I. Manajemen Event Tingkat Rendah

Jika terjadi interaksi antara pengguna dengan ponselnya maka terjadi suatu *event.* Misalnya jika pengguna memilih suatu menu maka sistem akan memproduksi sebuah *event* yang kemudian aplikasi akan diberitahu bahwa telah terjadi suatu *event* sebagai wujud interaksi dari pengguna. Seperti halnya pemrograman GUI, manajemen *event* ini juga terbagi atas dua level yakni level tinggi dan level rendah. Pada penulisan ini hanya akan dibahas manajamen tingkat rendah.

Untuk bisa menangani *event* pada level rendah, dibutuhkan *interface Canvas.* Karena kelas *Canvas* merupakan kelas abstrak sekaligus turunan dari kelas *Displayable.* Penggunaan kelas *Canvas* umumnya bersamaan dengan kelas *Graphics,* karena kelas *Graphics*-lah yang menyediakan objek-objek grafik dan fungsi-fungsi untuk yang

dibutuhkan untuk manipulasi grafik level rendah. Fungsi-fungsi yang disediakan oleh kelas Canvas adalah :

1. Proctected void keyPressed(int keyCode)

Fungsi ini digunakan untuk mendeteksi sebuah tombol ketika sedang ditekan.

2. Proctected void keyRelease(int keyCode)

Fungsi ini digunakan untuk mendeteksi sebuah tombol ketika selesai ditekan.

3. Proctected void keyRepeated(int keycode)

Fungsi ini digunakan untuk medeteksi sebuah tombol ketika ditekan terus menerus.

4. Proctected void pointerPressed(int x, int y)

5. Proctected void pointerDragged(int x, int y)

6. Proctected void pointerReleased(int x, int y)

J. Definisi Kunci Masukan pada Kelas Canvas

Pada kelas *canvas,* telah disediakan konstanta-konstanta yang penting untuk mengenali jenis masukan yang diberikan oleh pengguna. Konstanta ini merupakan nilai integer yang terdiri atas nilai-nilai berikut :

Tabel 4. Kostanta penting

No	Kostanta	Nilai Integer	Dikaitkan tombol angka
1	KEY_NUM 0	48	0
2	KEY_NUM 1	49	1
3	KEY_NUM 2	50	2
4	KEY_NUM 3	51	3
5	KEY_NUM 4	52	4
6	KEY_NUM 5	53	5
7	KEY_NUM 6	54	6
8	KEY_NUM 7	55	7
9	KEY_NUM 8	54	8

Tabel 4. lanjutan

10	KEY_NUM 9	54	9
11	KEY_STAR	42	*
12	KEY_POUND	35	#
13	UP	1	Panah atas
14	DOWN	6	Panah bawah
15	LEFT	2	Panah kiri
16	RIGHT	5	Panah kanan

Sebenarnya ada konstanta lain yang disediakan, yaitu Fungsi-fungsi pada kelas *Canvas* yang dikaitkan dengan nilai-nilai konstanta ini adalah :

1. Public int getGameAction

Akan menghasilkan nilai integer yang berkaitan dengan keyCode, yang umumnya diambil dari fungsi-fungsi keyPressed(), keyRealesed(), dan sejenisnya.

2. Public int String KeyName (int Key Code)

Akan menghasilkan nama kunci yang berkaitan dengan keyCode, yang umumnya diambil dari fungsi-fungsi keyPressed(), keyRealesed(), dan sejenisnya.

3. Public int getKeyCode (int gameAction)

Fungsi kebalikan dari getGameAction(), yang menghasilkan nilai integer keyCode yang berkaitan dengan nilai gameAction yang dihasilkan oleh fungsi getGameAction.

K. Pengenalan Teknologi *Wireless* dan Teori Aplikasi *Mobile*

Wireless dalam bahasa Indonesia disebut nirkabel**,** adalah teknologi yang menghubungkan dua piranti untuk bertukar data atau suara tanpa menggunakan media kabel. Data dipertukarkan melalui media gelombang cahaya tertentu (seperti teknologi

infra merah pada *remote* TV) atau gelombang radio (seperti *bluetooth* pada komputer dan ponsel) dengan frekuensi tertentu.

Dalam perkembangan *hardware* dan *software*, perkembangan teknologi yang begitu pesat adalah teknologi *wireless*. Teknologi *wireless* ini sendiri sangatlah luas mencakup bidang-bidang mulai radio, televisi, hingga peralatan komunikasi bergerak, seperti *pager*, *handphone*, maupun satelit. Secara konsep, teknologi *wireless* dapat dibagi menjadi dua kategori, pertama untuk lokal dan kedua untuk area yang luas. Untuk kategori pertama, peralatan yang termasuk di dalamnya, misalnya adalah *remote control* untuk mengunci atau pun membuka mobil maupun garasi, serta peralatan mainan dengan radio control, Dengan peralatan yang jenis pertama ini jangkauan yang dapat diterima tidaklah jauh. Sedangkan untuk jenis peralatan pada aplikasi yang kedua di antaranya adalah *pager*, *handphone.* Pada peralatan ini jangkauan yang dapat diterima adalah jauh lebih besar dari pada aplikasi yang pertama tadi. Jadi, sebuah peralatan komunikasi bergerak, seperti *handphone* menerima layanan dari sebuah *wireless carrier* atau perusahaan yang mengoperasikan *celltower* tersebut.

Pada awal perkembangan teknologi, masing-masing *vendor* menghasilkan *platform*, untuk aplikasi dan sistem operasinya sendiri, yang berhubungan dengan aplikasi komunikasi yang bergerak, Melihat perbedaan di atas ternyata sangat tidak menguntungkan bagi perkembangan komunikasi bergerak itu sendiri. Hal tersebut dikarenakan tidak fleksibelnya aplikasi yang dibuat untuk sebuah peralatan *handphone.* Ketika anda membuat sebuah aplikasi untuk sebuah peralatan *handphone*, anda membuat sebuah aplikasi yang sama yang harus dijalankan di *handphone* lain dan ternyata oleh

perbedaan *platform* aplikasi maupun sistem operasi, bisa jadi aplikasi tersebut tidak dapat dijalankan.

Hal ini tentu saja membawa kerugian terutama bagi para *developer*, karena membawa akibat yang sangat buruk bagi perkembangan aplikasi aplikasi baru. Karena alasan-alasan tersebut, standarisasi sangat perlu dilakukan. Selain standarisasi yang diperlukan untuk aplikasi-aplikasi tersebut, hal lain yang juga sama diperlukannya adalah sebuah bahasa pemograman yang memiliki kebebasan *platform* atau *platform independence*, dan ternyata Java memenuhi untuk alasan itu. Karena tujuan dari bahasa pemograman Java itu sendiri adalah *Write Once Run Anywhere*, dan untuk kepentingan peralatan komunikasi yang bergerak *Sun Microsystem* mengeluarkan edisi yang dinamakan J2ME atau *Java 2 MicroEdition.*

L. Rekayasa Perangkat Lunak

Rekayasa perangkat lunak adalah disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. Pada definisi ini, ada dua istilah kunci :

a. Disiplin rekayasa

   Perekayasa membuat suatu alat bekerja. Mereka menerapkan teori, metode, dan alat bantu yang sesuai, selain itu mereka menggunakannya dengan selektif dan selalu mencoba mencari solusi terhadap permasalahan, walaupun tidak ada teori atau metode yang mendukung. Perekayasa juga menyadari bahwa mereka harus bekerja

dalam batasan organisasi dan keuangan, sehingga mereka berusaha mencari solusi dalam batasan-bataan ini.

b. Semua aspek produksi perangkat lunak

Rekayasa perangkat lunak tidak hanya berhubungan dengan proses teknis dari pengembangan perangkat lunak tetapi juga dengan kegiatan seperti manajemen proyek perangkat lunak dan pengembangan alat bantu, metode dan teori untuk mendukung produksi perangkat lunak.

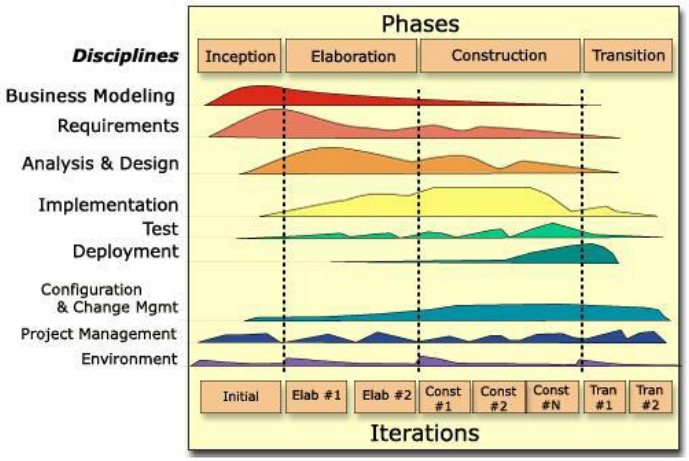
M. *Rational Unified Process* (RUP)

Metode pembangunan perangkat lunak merupakan panduan langkah demi langkah dalam membangun sebuah aplikasi perangkat lunak. Mengimplementasikan sebuah metode bertujuan agar dalam membangun sebuah aplikasi dapat lebih efisisen dan terencana, dengan mengikuti disiplin dari tiap tahapan yang ada di dalam metode tersebut. Metode-metode klasik seperti *waterfall life cyrcle* tidak bersifat fleksibel. Hal ini disebabkan oleh telah ditetapkannya batasan yang jelas antara kegiatan desain dengan implementasi suatu aplikasi. Kemungkinan perubahan pada setiap fase pasti ada dan tidak dapat dihindari. Oleh karena itu, diperlukan metode-metode baru yang bersifat terhadap perubahan *user requirement*. Perbaikan kesalahan-kesalahan yang ditemui dan perubahan-perubahan dari *user requirement* akan dapat segera dilakukan pada siklus berikutnya. Tahapan- tahapan ini bersifat iteratif dan inkremental. Beberapa metode pengembangan perangkat lunak yang termasuk dalam kriteria ini adalah XP (*Extreme Programming*), *Scrum*, FDD (*Feature Driven Development*), dan RUP (*Rational Unified Process*).

Pengembangan aplikasi permainan Puzzle ini akan menggunakan metode *Rational Unified Process* (RUP). Proses ini berorientasi pada fase, inkrementasi, dan proses iterasi. Tujuannya adalah untuk mendukung pengembangan kualitas suatu aplikasi secara tepat waktu dan sasaran (Rational the Software Development Company)

Terdapat dua pendekatan dalam metode RUP yaitu

1. Pendekatan secara inkremental (*incremental*) dilakukan dengan tujuan meningkatkan kualitas suatu produk., misalnya dengan adanya penambahan fitur dan fungsi pada suatu produk. Pada pendekatan ini, fungsi dasar aplikasi telah dipahami oleh *user*. Aspek penting dari *user requirement* telah dirancang, sebagai awal dari pengembangan suatu aplikasi.
2. Pendekatan secara iterasi direfleksikan ke RUP pada tiap fase, bertujuan melakukan proses perbaikan produk serta mengurangi resiko kesalahan. Di mana ke empat fase ini memiliki fokus pada proses pengembangan yang berbeda-beda. Masing-masing fase saling berhubungan satu sama lain. Adapun fase-fase tersebut adalah



Gambar 8. Diagram Fase RUP

a) Fase Permulaan (*Inception Phase*) adalah fase di mana pengembang mendefinisikan cakupan (*scope*) sistem secara global misalnya mengenai visi produk. Tujuannya adalah untuk mengembangkan aplikasi secara umum hingga ke tahap akhir. Hal ini meliputi definisi dari *user requirments*, contohnya pembatasan dari apa saja yang dapat dilakukan dan apa saja yang tidak harus dilakukan oleh sistem.

b) Fase Perluasan (*Elaboration Phase*) adalah fase di mana pengembang menganalisa kebutuhan aplikasi secara rinci dan mulai mendefinisikan pondasi arsitektur. Tujuan dari fase ini adalah mengurangi resiko kesalahan yang lebih banyak ketika mengembangkan aplikasi. Fase ini meliputi pemilihan arsitektur yang optimal dan stabil, tingkat pengembangan lebih lanjut, dan tingkat kemudahan penggunaan sistem.

c) Fase Konstruksi (*Construction Phase*) adalah fase di mana pengembang fokus pada melengkapi analisa, menampilkan gambaran desain secara global dan implementasi sistem. Pada fase ini telah dibangun suatu produk yang berhubungan dengan implementasi dari semua komponen dan integrasinya ke dalam suatu produk.

d) Fase Transisisi (*Transition Phase*) adalah fase di mana pengembang mengujicobakan sistem ke *use*r. Fase ini mengakhiri suatu proyek dengan melakukaan integrasi produk ke lingkungan *user*.

Tiap-tiap fase memiliki produk-produk yang dihasilkan dalam prosesnya, adapun produk-produk tersebut adalah :

1. Fase Insepsi

a) Membuat visi yang berisi tujuan akhir sistem

Kerangka utama dari fase insepsi adalah visi. Visi dapat direpresentasikan sebagai dokumen formal maupun informal, sebagai catatan hasil komunikasi antara *customer* dengan pihak atau tim pengembang sistem. Tujuan visi adalah agar terdapat kesamaan pandangan antara pihak *customer* dengan tim pengembang dalam hal tujuan utama dibangunnya suatu aplikasi.

b) Membuat batasan-batasan atau *scope* sistem

Scope berisi batasan-batasan kerja dari sistem, fungsi-fungsi utama apa saja yang dapat dilakukan sistem. Penentuan *scope* ini berguna untuk memberi batasan fitur-fitur utama yang harus ada pada sistem.

c) Membuat *general use case*

General uses case berisi gambaran *use case* dari keseluruhan aktor secara umum. Hal ini sekaligus menunjukkan hubungan antara aktor satu dengan yang lainnya, berdasarkan dari adanya kesamaan *use case* yang digunakan.

d) Identifikasi resiko

Dalam pengembangan sebuah perangkat lunak, resiko adalah faktor utama yang harus diperhitungkan. Tidak adanya identifikasi resiko sedari awal akan menyebabkan kegagalan dalam pembangunan sebuah sistem.

e) Membangun lingkungan yang mendukung dan infrastruktur yang digunakan. Meliputi pemilihan *software*, instalasi, dan konfigurasi *tools*. Kemudian melakukan pengujian terhadap *tools* tersebut apakah sudah dikonfigurasi dengan benar.

f) Menentukan arsitektur sistem

Tahap ini adalah yang paling penting dalam proses pembangunan sistem. Arsitektur sisitem yang stabil akan mengurangi terjadinya resiko perubahan yang cukup besar dan proses *maintenance* yang lebih mudah. Oleh karena itu, diperlukan analisis yang teliti dalam menentukan arsitektur yang cocok dalam membangun sebuah sistem.

g) Iterasi dan Inkrementasi

Pada tahap ini akan dilakukan evaluasi dari produk-produk yang dihasilkan pada fase insepsi. Proses ini meliputi perbaikan produk dan penambahan fitur-fitur yang dianggap perlu.

2. Fase Elaborasi

a) Iterasi fase sebelumnya

Berupa *update* produk yang merupakan lanjutan dari tahap iterasi dan inkrementasi pada fase insepsi.

b) Menentukan *detail use case*

Tahap ini merupakan penentuan *use case* yang stabil dan lebih detail dari masing-masing aktor dalam sistem berupa diagram *use case* dari tiap-tiap aktor.

c) Skenario *use case*

Skenario *use case* berisi gambaran *use case* secara detail yang meliputi nama *use case*, deskripsi *use case*, aktor, alur kejadian dan hubungan antar *use case* satu dengan yang lainnya.

d) Desain *class* diagram

Menggambarkan struktur dan deskripsi *class, package,* dan objek beserta hubungan satu sama lain. Pada desain diagram kelas ini, ditentukan *entity* yang akan digunakan oleh sistem, atribut dan *method* tiap kelas, beserta relasi antar kelas.

e) Desain *sequence* diagram

Menggambarkan interaksi antar objek di dalam dan sekitar sistem atau menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah kejadian untuk menghasilkan keluaran tertentu.

f) Desain *activity* diagram

Menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, keputusan yang mungkin terjadi, dan bagaimana hasil keputusan yang didapatkan.

g) Implementasi desain *User Interface*

Mengimplementasikan hasil desain tampilan halaman aplikasi yang terdapat dalam pemodelan presentasi.

h) Iterasi dan Inkrementasi

Pada tahap ini akan dilakukan evaluasi dari produk-produk yang dihasilkan pada fase elaborasi. Proses ini meliputi perbaikan produk dan penambahan fitur-fitur yang dianggap perlu.

3. Fase Konstruksi

a) Memperbaiki dan melengkapi arsitektur dan model implementasi

Berupa perbaikan desain *database*, kelas diagram, desain *user interface*, dan segala hal yang berhubungan dengan arsitektur sistem.

b) Melakukan pengkodean secara bertahap

Pada tahap ini dilakukan pengkodean pada setiap unit secara bertahap dalam bentuk paket dari tiap *entity*.

c) *Testing use case*

Pada tahap ini dilakukan *testing* oleh seorang *tester* pada tiap-tiap *use case* berdasarkan skenario *use case* yang dibuat. Hal ini untuk menjamin apakah masing-masing *use case* dapat berfungsi dengan baik, apakah *use case* telah sesuai dengan aktor yang bersangkutan, apakah tampilan *user interface* telah sesuai, apakah navigasi telah sesuai, dan berbagai hal yang berhubungan dengan *layer database, business logic*, dan presentasi.

d) *Testing beta product*

Pada tahap ini sebagian besar *use case* telah berfungsi dengan baik dan sesuai dengan *user requirement*. Sistem telah mencakup visi dan *scope* yang ada. Namun masih terdapat beberapa *bugs* atau hal-hal lain yang masih perlu diperbaiki. *Beta product* masih memiliki kemungkinan terjadinya perbaikan dan penambahan, namun tidak mengubah arsitektur sistem secara signifikan.

4. Fase Transisi

a) *Tuning code* dan perbaikan kesalahan

Pada tahap ini dilakukan perbaikan-perbaikan sistem dari hasil *testing beta product.*

b) *Final release*

Pada tahap ini dilakukan *testing* sistem secara keseluruhan oleh seorang *tester*. Sistem dianggap sudah siap untuk diintegrasikan ke lingkungan *user*.

c) Integrasi produk ke lingkungan *end-user*

Apabila sistem sudah stabil dan sesuai dengan permintaan *user*, maka dilakukan integrasi sistem ke lingkungan *end-user*.