BAB II

LANDASAN TEORI

2.1 Pengertian Aplikasi

Aplikasi merupakan penerapan pengimplementasian suatu hal atau data permasalahan dan pekerjaan dalam suatu sarana atau media yang dapat digunakan untuk menerapakan atau mengimplementasikan permasalahan tersebut. Sehingga berubah bentuk menjadi bentuk yang baru tanpa menghilangkan nilai dasar dari hal atau data permasalahan dan pekerjaan. Jadi dalam hal ini hanya bentuk dari tampilan data yang berubah, sedangkan isi yang termuat dalam data tersebut tidak mengalami perubahan. [3]

2.2 Pengertian Kamus

Kamus merupakan salah satu jenis buku yang dipergunakan manusia sebagai alat pembelajaran menguasai bahasa, dimana kamus dapat dipergunakan sebagai alat penterjemah dari satu bahasa ke bahasa lainnya. [2]

2.3 Kamus Inggris-Indonesia dan Indonesia-Inggris

Kamus Inggris-Indonesia dan Indonesia-Inggris merupakan usaha pendaftaran yang luas yang bertujuan mencakup sebagian besar kata dan ungkapan bahasa Inggris yang paling umum dipakai di Amerika, serta bahasa Indonesia yang secara resmi dipakai di Indonesia. [1]

Pada kamus ini mengandung terjemahan dan keterangan kata dari bahasa Inggris ke bahasa Indonesia maupun sebaliknya. Secara umum kata didalam bahasa Inggris dapat dibedakan sebagai berikut :

a. *Verb* (Kata Kerja)

Verb menyatakan apa yang dilakukan oleh subjek, atau apa yang terjadi pada subjek. *Verb* tersebut dapat digunakan untuk membuat pernyataan, menanyakan sebuah pertanyaan, atau memberikan perintah.

b. *Articles* (Kata Sandang)

Dalam bahasa Inggris, *articles* (kata sandang) dapat dibedakan dalam dua macam, yaitu :

1) *Definite Article* (Kata Sandang Tertentu), adalah kata yang dipakai di depan kata benda tunggal atau jamak, dimana benda tersebut sudah diketahui sebelumnya. Kata yang dipakai adalah "*the*".
2) *Indefinite Article* (Kata Sandang Tak Tentu), adalah kata yang dipakai di depan kata benda tunggal atau jamak, dimana benda tersebut belum diketahui sebelumnya. Kata yang dipakai adalah kata sandang "*a*" atau "*an*".

c. *Auxiliary Verbs* (Kata Kerja Bantu)

Auxiliary verbs atau sering juga disebut *helping verb* (kata kerja bantu) dalam bahasa Inggris terdapat kurang-lebih 24 macam. Dari ke-24 macam *auxiliary verb* ini ada beberapa di antaranya yang berfungsi sebagai kata kerja, seperti : *do, does, did, have, has*, dan lain-lain.

d. *Preposition* (Kata Depan)

Preposition (kata depan) adalah kata yang diletakkan di depan atau sebelum kata benda yang berfungsi untuk menerangkan letak atau posisi benda atau kata yang lainnya.

e. *Nouns* (Kata Benda)

Dalam bahasa Inggris, *nouns* (kata benda) pada umumnya dibagi menjadi dua macam, yaitu :

1) *Concrete Nouns* (Kata Benda Konkrit), adalah kata yang menyatakan mengenai benda-benda yang berwujud (nyata).
2) *Abstract Nouns* (Kata Benda Abstrak), adalah kata yang menyatakan mengenai benda-benda yang tidak berwujud, sehingga benda-benda tersebut tidak dapat dilihat dengan nyata.

f. *Adjectives* (Kata Sifat)

Adjective (kata sifat) adalah suatu kata yang digunakan untuk menerangkan kata benda (*nouns*).

g. *Adverbs* (Kata Keterangan)

Adverbs (kata keterangan) adalah suatu kata yang menerangkan kata kerja, kata sifat, atau kata keterangan lainnya, selain kata benda.

2.4 Pemograman Berorientasi Objek (PBO)

OOP (*Object Oriented Programming*) atau disebut juga dengan PBO (Pemograman Berorientasi Objek) merupakan suatu kelas bahasa pemrograman dan teknik yang didasarkan pada suatu objek, dimana struktur data terenkapsulasi dengan rutin yang mengoperasikan data tersebut. [7]

Keunggulan utama pemrograman berorientasi objek adalah kemampuannya untuk mengembangkan objek-objek sekali saja dan menggunakannya lagi (*reusable*) berulang-ulang.

Terdapat Karakteristik-karakteristik dalam Pemograman Berorientasi Objek yaitu :

a. Pembungkusan (*Encapsulation*)

Dalam sistem berorientasi objek kita menggabungkan potongan-potongan informasi dan perilaku-perilaku spesifik yang bekerja pada informasi tersebut, kemudian mengemasnya menjadi apa yang disebut sebagai objek. Hal ini dirujuk dari kata pembungkusan (*encapsulation*). Cara lain umtuk memahami pembungkusan adalah kita membagi aplikasi-aplikasi menjadi bagian-bagian kecil yang secara fungsional berhubungan.

b. Pewarisan (*Inheritance*)

Pewarisan adalah konsep dasar pemograman berorientasi objek yang lainnya. Pada system berorientasi objek, pewarisan adalah mekanisme yang memungkinkan kita menciptakan objek-objek baru berdasarkan objek lain yang sudah ada: Objek *Anak* mewarisi segala sesuatunya dari objek *Induk*.

Salah satu keuntungan dari pewarisan (*inheritance*) adalah kemudahannya untuk dipelihara. Saat sesuatu perubahan untuk informasi atau perilaku yang sama perlu terjadi pada seluruh objek.

c. *Polimorfisme*

Polimorfisme adalah prinsip penting yang ketiga dari pemograman berorientasi objek. *Polimorfisme* berarti suatu fungsionalitas yang diimplementasikan dengan berbagai cara yang berbeda. Pada terminologi

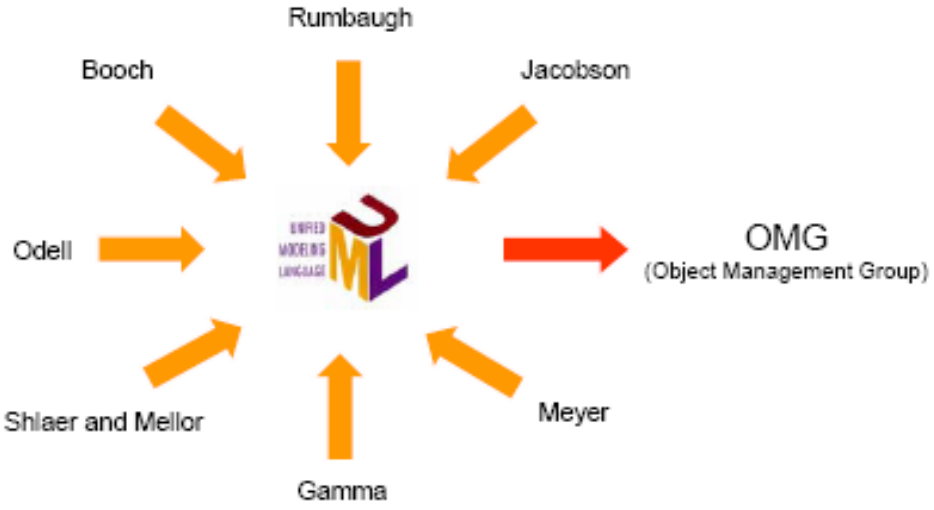
berorientasi objek, ini berarti kita dapat memiliki berbagai implementasi untuk sebagian fungsionalitas tertentu. Salah satu keuntungan dari *polimorfisme* adalah kemudahan pemeliharaannya.

2.5 UML (*Unified Modeling Language*)

*Unified Modelling Langu*age (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. [7]

2.5.1 Sejarah UML

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.



Gambar 2.1 Sejarah UML

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

2.5.2 Tujuan Penggunaan UML

Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi *object*. Dan menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.

2.5.3 Gambaran dari UML

UML dapat dimanfaatkan untuk berbagai macam kebutuhan, diantaranya untuk melakukan pemodelan, visualisasi, spesifikasi, konstruksi dan dokumentasi.

2.5.4 Notasi dalam UML

UML menyediakan beberapa notasi dan artifak standar yang bisa digunakan sebagai alat komunikasi bagi para pelaku dalam proses analisis dan desain. Artifak didalam UML didefinisikan sebagai informasi dalam berbagai bentuk yang digunakan atau dihasilkan dalam proses pengembangan perangkat lunak. Contohnya adalah *sourcecode* yang dihasilkan oleh proses pemograman.

Yang perlu diperhatikan untuk menjaga konsistensi antar artifak selama proses analisis dan desain adalah bahwa setiap perubahan yang terjadi pada satu artifak harus dilakukan pada artifak sebelumnya. Misalnya ditemukan satu cara yang lebih efisien waktu membuat *Sequence* diagram, maka perbaikan tersebut perlu diverifikasikan terhadap *Use Case* diagram, apakah perlu dilakukan perubahan juga pada *Use Case* spesifikasinya.

a. Actor

Actor adalah segala sesuatu yang berinteraksi dengan sistem aplikasi komputer. Jadi actor ini bisa berupa orang, perangkat keras, atau mungkin objek lain dalam sistem yang sama. Biasanya yang dilakukan oleh *actor* adalah memberikan informasi pada sistem atau memerintahkan sistem untuk melakukan sesuatu.

b. Class

Class merupakan pembentuk utama dari sistem berorientasi objek karena class menunjukkan kumpulan objek yang memiliki atribut dan operasi yang sama. *Class* digunakan untuk mengimplementasikan *interface. Class* digunakan untuk mengabstraksikan elemen-elemen dari sistem yang sedang dibangun. *Class* bisa untuk merepresentasikan baik perangkat lunak maupun perangkat keras, baik konsep maupun benda nyata.

Notasi class berbentuk persegi panjang berisi tiga bagian yaitu :

1) Persegi panjang atas untuk nama class;
2) Persegi panjang bawah untuk operasi;
3) Persegi panjang tengah untuk atribut.

Atribut digunakan untuk menyimpan informasi. Nama atribut menggunakan kata benda yang bisa dengan jelas merepresentasikan informasiyang disimpan didalamnya. Operasi menunjukkan sesuatu yang bisa dilakukan oleh objek, dan menggunakan kata kerja.

c. Interface

Interface merupakan kumpulan operasi tanpa implementasi dari suatu *class*. Implementasi operasi dalam interface dijabarkan oleh operasi dalam *class*. Oleh karena itu, keberadaan interface selalu disertai oleh *class* yang mengimplementasikan operasinya. Interface ini merupakan salah satu cara mewujudkan prinsip enkapsulasi dalam objek.

d. Use Case

Use case menjelaskan urutan kegiatan yang dilakukan *actor* dan sistem untuk mencapai suatu tujuan tertentu. Walaupun menjelaskan kegiatan namun

use case hanya menjelaskan apa yang dilakukan oleh *actor* dan sistem, bukan bagaimana *actor* dan sistem melakukan kegiatan tersebut.

e. Interaction

Interaction digunakan untuk menunjukkan aliran pesan atau informasi antar objek maupun hubungan antar objek. Biasanya *interaction* ini dilengkapi juga dengan teks bernama operation signature yang tersusun dari nama operasi, parameter yang dikirim dan tipe parameter yang dikembalikan.

f. Package

Package adalah kontainer atau wadah konseptual yang digunakan untuk mengelompokkan elemen-elemen dari sistem yang sedang dibangun, sehingga bisa dibuat model yang lebih sederhana. Tujuannya adalah untuk mempermudah penglihatan (*visibility*) dari model yang sudah dibangun.

g. Note

Note digunakan untuk memberikan keterangan dan komentar tambahan dari suatu elemen sehingga bisa langsung terlampir dalam model. *Note* ini bisa ditempatkan kesemua elemen notasi yang lain.

h. Dependency

Dependency merupakan relasi yang menunjukkan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Elemen yang ada di bagian tanda panah adalah elemen yang tergantung pada elemen yang ada dibagian tanda panah.

Terdapat dua *stereotype* dari dependency, yaitu *Include* dan *Extend*. *Include* menunjukkan bahwa suatu bagian dari elemen (yang ada digaris tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada digaris dengan

panah), misalnya untuk notasi A - - - > B operasi yang ada di class A memicu di eksekusinya operasi yang ada di class B. Extended menunjukkan bahwa suatu bagian elemen digaris tanpa panah bisa disisipkan kedalam elemen yang ada digaris dengan panah, misalnya untuk notasi A - - - > B suatu fungsi dari use case A bisa disisipkan kedalam use case B atau dengan kata lain A optional untuk B.

Kedua *stereotype* ini direpresentasikan dengan menambahkan *text include* atau *extends* di notasi *dependency*.

i. Boundary Class

Boundary Class adalah *class* yang menghubungkan *user* dengan sistem. Oleh karena itu sering disebut juga sebagai *user interface class*.

j. Control Class

Control Class adalah *class* yang mengkoordinasi aktivitas dalam sistem. *Class* ini menghubungkan *boundary class* dengan *entity class*.

k. Entity Class

Entity class adalah *class* yang berhubungan dengan data atau informasi yang digunakan oleh sistem. *Entity class* ini adalah *class* yang menyimpan dan mengelola data.

2.5.5 Diagram Grafis

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk v*iew* tertentu. Adapun jenis diagram antara lain :

a. Use Case Diagram

Menggambarkan sejumlah *external actors* dan hubungannya ke *use case* yang diberikan oleh sistem. Use case adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari use case symbol namun dapat juga dilakukan dalam *activity diagrams*. *Use case* digambarkan hanya yang dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat user) dan bukan bagaimana fungsi yang ada di dalam sistem.

b. Class Diagram

Menggambarkan struktur statis *class* di dalam sistem. *Class* merepresentasikan sesuatu yang ditangani oleh sistem.

Class dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu class tergantung/menggunakan class yang lain), *specialed* (satu class merupakan spesialisasi dari class lainnya), atau *package* (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa *class diagram*.

c. State Diagram

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

d. Activity Diagram

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

e. Sequence Diagram

Menggambarkan kolaborasi dinamis antara sejumlah *object*. Kegunaanya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

f. Collaboration Diagram

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

g. Component Diagram

Menggambarkan struktur fisik kode dari komponent. Komponent dapat berupa *source code*, komponent *biner*, atau *executable component*. Sebuah komponent berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*.

h. Deployment Diagram

Menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh node tertentu dan ketergantungan komponen.

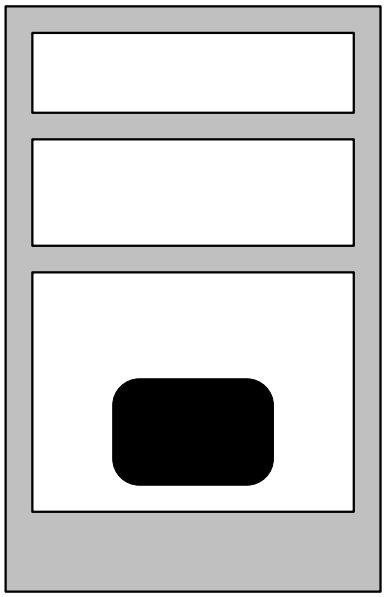
2.6 Pengenalan J2ME

Sun memperkenalkan sebuah bentuk bahasa Java yang dioptimalkan yang dinamai J2ME. Platform J2ME ini diarahkan untuk piranti kecil dan mobile seperti ponsel dan PDA. Sekarang ini, kebanyakan produsen ponsel dan PDA mempunyai komitmen yang tinggi untuk memasukkan Java ke dalam piranti tersebut. Walaupun J2ME bukan satu-satunya bahasa yang digunakan pada ponsel, namun standar industry telah menggunakan Java sebagai salah satu standar yang perlu ada pada ponsel. [9]

2.7 Pengertian J2ME

J2ME merupakan sebuah kombinasi yang terbentuk antara sekumpulan *interface* java yang sering disebut dengan java API (*Application Programming Interface*) dengan JVM yang didesain khusus untuk alat, yaitu JVM dengan ruang yang terbatas. Kombinasi tersebut kemudian digunakan untuk melakukan pembuatan aplikasi-aplikasi yang dapat diintegrasikan kedalam piranti elektronik (dalam hal ini *mobile device*). [8]

J2ME terdiri dari 3 buah bagian, yaitu : konfigurasi, profil dan paket-paket opsional.



Gambar 2.2 Bagian-bagian Didalam *Platform* J2ME

a. Konfigurasi

Konfigurasi merupakan bagian yang berisi JVM dan beberapa *library* kelas lainnya. Terdapat 2 buah konfigurasi yang disediakan oleh *Sun Microsystems* yaitu CLDC (*Connected Limited Device Configuration*) dan CDC (*Connected Device Configuration*). Target alat dari konfigurasi CLDC adalah alat-alat kecil, seperti telepon seluler. Dimana CLDC (*Connected Limited Device Configuration*) 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.

b. Profil

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

Adapun profil yang sangat populer penggunaannya adalah *profile* yang disediakan oleh Sun Microsystems yang dinamai MIDP (*Mobile Information Device Profile*).

c. Paket-paket Opsional

Paket-paket opsional merupakan paket-paket tambahan yang dibutuhkan oleh aplikasi, sehingga pada saat proses *deployment* paket-paket tersebut perlu didistribusikan juga sebagai bagian dari aplikasi bersangkutan.

2.7.1 MIDlet

MIDlet adalah aplikasi yang ditulis untuk 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.

2.7.2 RMS (*Record Management System*)

MIDlet tidak menggunakan *file system* untuk menyimpan data, tetapi menyimpan semua informasi dalam sebuah memori *non-volatile* (memori tetap) yang disebut dengan *Record Management System* (RMS). [8]

RMS (*Record Management System*) adalah kumpulan *record*, dan *record* disimpan sebagai *array byte* dari data dalam sebuah *record store*. RMS memiliki orientasi *record* basis data yang sederhana sehingga MIDlet dapat menyimpan informasi dan mengaksesnya. MIDlet yang berbeda dapat mengakses RMS yang sama.

Proses komputasi biasanya tidak maksimal tanpa dukungan kemampuan untuk menyimpan data dan mengambil data. Meskipun aplikasi sangat sederhana, sering dibutuhkan suatu cara untuk menjaga agar data yang diproses bisa disimpan pada perangkat mobile.

Walaupun RMS dapat melakukan apa yang dapat dilakukan oleh database, tetapi RMS bukan suatu database relasional dan tidak dapat menggunakan perintah-perintah SQL untuk berinteraksi dengan data.

Kemampuan RMS merupakan *persistent storage* yaitu tempat penyimpanan objek yang tidak mudah hilang. Untuk beberapa aplikasi, kita mungkin membutuhkan suatu data tetap tetap bertahan walaupun kita sudah keluar dari aplikasi yang dipakai. Tanpa *persistent storage*, objek dan semua data akan rusak/hilang ketika kita menutup aplikasi. Ini berarti, jika kita menggunakan *persistent storage* waktu hidup data akan lebih lama daripada program dan jika kita menggunakan program ini lagi maka data tersebut masih tetap ada (bertahan). [8]

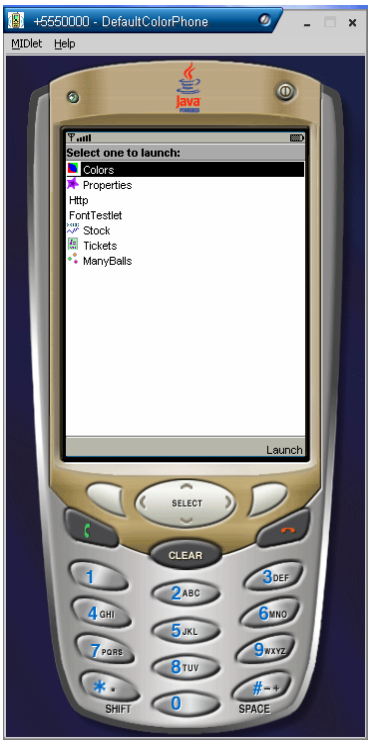
2.7.3 Menyeleksi *Record*

RMS menyediakan *class* abstrak dalam mengakses *record* untuk mode pencarian *record*. RecordFilter adalah *class* abstrak dari RMS yang biasa

dipergunakan untuk proses pencarian *record*. RecordFilter memiliki metode yang harus diimplementasikan yaitu `boolean matches ()`, metode ini mengembalikan nilai *true* jika data yang dicari ditemukan dalam proses pencarian, atau nilai *false* jika data yang dicari tidak ditemukan. [9]

2.7.3 J2ME WTK (*Wireless Tool Kit*)

J2ME WTK adalah program yang meniru kerja ponsel yang mendukung MIDP atau yang biasa disebut dengan emulator. Untuk menjalankan aplikasi ponsel, programmer tidak perlu memiliki dan mencobanya pada ponsel. Cukup dengan emulator dari ponsel yang dapat dijalankan pada PC. Berikut adalah emulator yang disediakan oleh J2ME Wireless Toolkit :



Gambar 2.3 Emulator J2ME *Default Color Phone*



Gambar 2.4 Emulator Sony Ericson Dengan Skin SE W550

2.8 Metode Pencarian Beruntun

Pada dasarnya, metode pencarian beruntun adalah proses membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama, sampai elemen yang dicari ditemukan, atau seluruh elemen sudah diperiksa. [6]

2.8.1 Metode Pencarian Beruntun Tanpa Boolean

Metode pencarian ini dalam melakukan proses pencarian datanya tanpa menggunakan tipe data *boolean*. Terdapat dua varian pada metode pencarian beruntun tanpa boolean, yaitu :

a) Metode Pencarian Beruntun tanpa Sentinel

Membandingkan elemen pertama dari array dengan data yang dicari, jika tidak sama, maka pencarian dilakukan ke elemen berikutnya. Lakukan proses tersebut sampai data ditemukan atau sampai indeks array lebih besar atau sama dengan maksimum data / elemen. Jika data ditemukan maka tampilkan posisi/indeksnya, dan jika data tidak ditemukan maka beri pesan.

b) Metode Pencarian Beruntun dengan Sentinel

Jika pencarian bertujuan untuk menambahkan elemen baru setelah elemen terakhir larik, maka terdapat sebuah varian dari metode pencarian beruntun yang mangkus.

2.8.2 Metode Pencarian Beruntun Dengan Boolean

Variabel bertipe boolean digunakan pada pencarian untuk menentukan apakah data yang dicari ditemukan atau tidak, jika data yag dicari ditemukan maka nilai variabel bertipe boolean adalah *true,* tapi jika tidak ditemukan bernilai *false*. Pencarian ini lebih cepat dibandingkan dengan metode sekuensial/beruntun lainnya.

Setiap elemen larik L dibandingkan dengan x mulai dari elemen pertama, L (1). Aksi pembandingan dilakukan selama indeks larik *i* belum melebihi *n* dan L(i) tidak sama dengan x. Aksi pembandingan dihentikan bila L(i) = x atau i = n. Elemen terakhir, L(n), diperiksa secara khusus. Keluaran yang dihasilkan adalah bernilai *true* jika x ditemukan, atau bernilai *false* jika x tidak ditemukan.

Metode pencarian beruntun dengan boolean dapat dituliskan sebagai berikut :

(i) Prosedure Pencarian Beruntun

```
procedure SeqSearch (input L : larikInt, input n : integer,
                     input x : integer, output ketemu: boolean)
{ Mencari keberadaan nilasi x didalam larik L (1..n]. }
{ K.Awal  : x dan larik L[1..n] sudah terdefinisi nilainya. }
{ K.Akhir : ketemu bernilai true jika x ditemukan. Jika x tidak
  ditemukan, ketemu bernilai false. }

DEKLARASI
   i : integer                    {pencatat indeks larik}


ALGORITMA :
   i ← 1
   while (i < n) and (L{i}≠x) do
      i ← i + 1
   endwhile
   {i = n or L(i)≠x }

   if L(i) = x then                { x ditemukan }
      ketemu ← true
   else
      ketemu ← false               { x tidak ada didalam larik L }
   endif
```

(i) Fungsi Pencarian Beruntun

```
function SeqSearch (input L : larikInt, input n : integer,
                    input x : integer) → boolean
{ Mengembalikan nilai true jika x ditemukan dalam larik L (1..n],
atau nilai false jika x tidak ditemukan. }

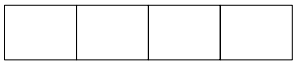
DEKLARASI
   i : integer                    {pencatat indeks larik}

ALGORITMA
   i ← 1
   while (i < n) and (L{i}≠x) do
      i ← i + 1
   endwhile
   {i = n or L(i)≠x }

   if L(i) = x then                { x ditemukan }
      ketemu ← true
   else
      ketemu ← false               { x tidak ada didalam larik L }
   endif
```

{ Pengujian Algoritma }

Misal :



Data yang dicari = 3

i = 1
Ap (1 < 4) and (L(1) ≠ x)
T T = T
Ap (L(1) = datacari) ?
(6 = 3) ? F, maka

i = 2
Ap (2 < 4) and (L(2) ≠ x)
T T = T
Ap (A(2) = datacari) ?
(3 = 3) ? T, maka
Ketemu = true
Ap (2 < 4) and (L(2) x)
T F = F
{keluar dari pencarian}
Ap (ketemu) ? True, maka
Data 3 ketemu pada posisi ke-2

Misal : data yang dicari = 9

i = 1
Ap (1 < 4) and (L(1) ≠ x)
T T = T
Ap (A(1) = datacari) ?
(6 = 9) ? F, maka

i = 2
Ap (2 < 4) and (L(2) ≠ x)
T T = T
Ap (A(2) = datacari) ?
(3 = 9) ? F, maka

i = 3
Ap (3 < 4) and (L(3) ≠ x)
T T = T
Ap (A(3) = datacari) ?
(1 = 9) ? F, maka

i = 4
Ap (4 < 4) and (L(4) ≠ x)
F T = F
{keluar proses pencarian}
Ap (ketemu) ? False, maka
Data 9 tidak ditemukan