# 2. TEORI DASAR

## 2.1. Teknologi Java

Bahasa Java awalnya dikembangkan oleh Sun Microsystem untuk melakukan pemrograman pada perangkat elektronik rumah tangga seperti televisi dan radio. Namun dalam perkembangannya bahasa Oak, nama awal bahasa ini, menjadi bahasa pemrograman yang digunakan sebagai bahasa pemrograman secara umum (yang saat ini dikenal dengan nama Java). Bahasa Java ini memiliki slogan *"Write Once Run Anywhere"* yang artinya cukup menuliskan sekali saja *source code* Java, aplikasi tersebut dapat dijalankan di semua *platform* yang mendukung Java *Virtual Machine*. (Wicaksono, 2002, p.2)

Salah satu teknologi Java yang ditawarkan adalah *write once run anywhere*. Java sebagai bahasa pemrograman dikenal sebagai bahasa pemrograman tingkat tinggi dengan fitur utamanya, yaitu :

- Keamanan (*security*)

  Java dirancang sebagai bahasa pemrograman yang aman (*secure*) karena suatu aplikasi yang menggunakan bahasa Java harus dieksekusi menggunakan Java *Virtual Machine* (JVM) yang menyediakan lingkungan yang aman untuk eksekusi kode yang telah di-*download*. Dalam keadaan yang paling buruk sekalipun tidak akan mengacau sebuah peralatan namun hanya mengacaukan *Virtual Machine*-nya saja.

- *Object Oriented*

  Paradigma pemrograman berorientasi *object* merupakan paradigma alami untuk membuat program yang dapat bertahan terhadap perubahan yang berkaitan dengan pertumbuhan dan perkembangan suatu sistem dan Java sebagai bahasa pemrograman berorientasi objek. (Naughton, 2002, p.21).

- Kokoh (*robust*)

  Bahasa pemrograman Java adalah dapat dikatakan kokoh karena penggunaan *garbage collector* yang dapat meningkatkan efisiensi waktu yang diperlukan untuk mencari kebocoran dalam penggunaan *memory*. Salah satu kendala keamanan pada bahasa C/C++ yaitu *buffer overflow*

dimana adanya kegagalan manajemen *memory*, tidak terjadi di Java karena memiliki pengaturan *memory* yang lebih baik. Selain itu mekanisme *exception* meningkatkan kemampuan *programmer* dalam membuat suatu aplikasi Java yang kokoh. (Hartanto, 2003, p.2-3).

- Kompatibilitas dengan Berbagai Arsitektur

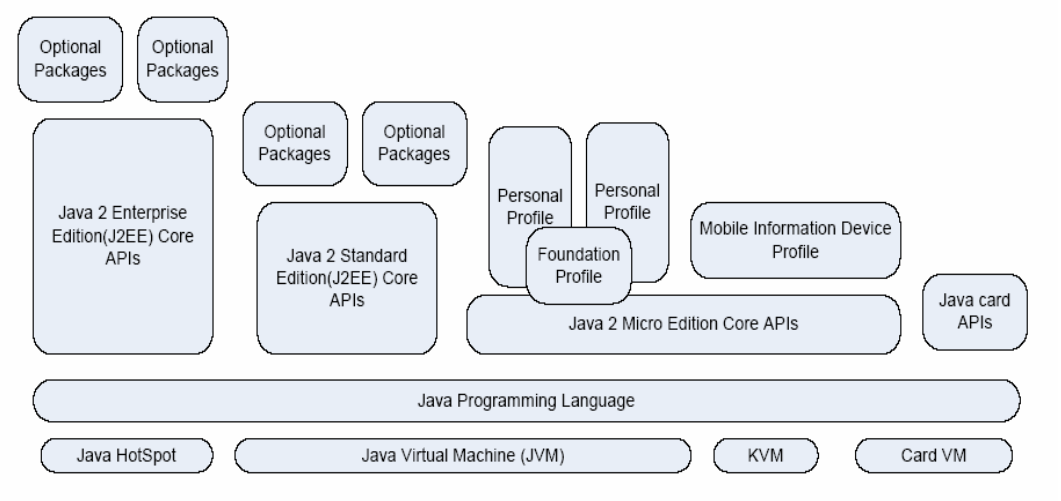
  Dengan adanya slogan "*Write Once Run Anywhere*", Java dirancang untuk dapat dijalankan di semua *platform* tanpa membedakan arsitektur perangkat lunak dan keras yang digunakan. Dalam hal ini aplikasi Java mampu berjalan pada semua *platform* tanpa harus ada kompilasi ulang. Hal ini disebabkan karena Java dirancang untuk menghasilkan kode yang netral terhadap semua *platform* perangkat keras dan lunak yang digunakan yang biasa disebut dengan *Javabyte code*.

- Interpretasi dan Berkinerja Tinggi

  Java dirancang untuk menghasilkan aplikasi-aplikasi dengan *performance* terbaik sekalipun bekerja di komputer yang tidak terlalu kuat.Walaupun Java merupakan bahasa interpretasi, kode-kode Java telah dibuat dengan hati-hati agar mudah diterjemahkan ke dalam bahasa asli suatu mesin agar mendapatkan kinerja yang tinggi.

Pada awal mula perkembangannya, Sun Microsystem meluncurkan produksi pertama dari Java dengan penggunaan Java *Development Kit* (JDK) 1.0.2. JDK. Sun Microsystem kemudian merilis JDK terbaru versi 1.1. Pada JDK versi 1.1 dibagi menjadi 2 bagian yaitu JRE (Java *Runtime Enviroment*) yang dikhususkan menjalankan program Java dan JSDK (Java *Software Development Kit*) yang terdiri atas paket-paket yang dapat digunakan untuk mengkompilasi sekaligus menjalankan program-program Java. Aplikasi yang kompatibel dengan JDK dan JRE sampai dengan versi 1.1 ini kemudian dikenal dengan nama Java 1 *Compliant.*

Pada perkembangan Java selanjutnya, Sun Microsystem Java 2 (terdiri atas JDK 1.2 dan JRE 1.2), dikenal dengan Java 2 *Compliant*. Adapun teknologi Java 2, digambarkan pada Gambar 2.1.




Gambar 2.1. Lingkungan Kerja Teknologi Java

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.5

Secara garis besar, teknologi Java 2 dibagi menjadi 3 yaitu :

1. Java 2 *Standard Edition* (J2SE)

   Kategori pertama dari Java ini digunakan unutk menjalankan aplikasi dan mengembangkan aplikasi Java pada lingkungan *Personal Computer* (PC).

2. Java 2 *Enterprise Edition* (J2EE)

   Kategori ini digunakan pada perangkat keras yang mempunyai spesifikasi dan *memory* yang besar seperti pada komputer *server* untuk menjalankan dan mengembangkan aplikasi-aplikasi Java di lingkungan *enterprise* dengan penambahan dukungan fungsionalitas Java seperti EJB (*Enterprise Java Bean*), Java CORBA, Servlet, Java *Server* Page (JSP) serta Java XML (*Extensible Markup Language*).

3. Java 2 *Micro Edition* (J2ME)

   Kategori terakhir dari Java ini digunakan untuk mengembangkan aplikasi Java pada *handheld devices* yang memiliki spesifikasi memori yang terbatas, seperti perangkat *handphone*, *Palm*, *pager*, *Personal Digital Assistance*(PDA), *Pocket PC* dan sejenisnya. (Wicaksono, 2002, p.4-5).

## 2.2. Java 2 Micro Edition (J2ME)

Telah disebutkan sebelumnya bahwa teknologi Java sendiri terbagi menjadi beberapa macam. Untuk kepentingan pengembangan aplikasi pada

perangkat keras dengan memori dan kapasitas penyimpanan yang terbatas, digunakan J2ME. Teknologi berbeda dengan teknologi Java yang lainnya, karena karakteristik perangkatnya yang digunakan untuk menjalankan juga berbeda.

2.2.1. Karakteristik J2ME

Seperti telah disebutkan sebelumnya bahwa J2ME khusus dirancang pada perangkat yang memiliki karakteristik memori terbatas seperti *handphone*, karena jelas tidak mungkin menggunakan program yang dirancang untuk dijalankan pada komputer lalu dijalankan di perangkat dengan *memory* yang sangat terbatas. Adapun komponen-komponen J2ME terdiri dari:

a. Java *Virtual Machine* (JVM)

   Komponen ini digunakan untuk menjalankan program-program Java pada *handheld devices* atau *emulator*.

b. Java API (*Application Programming Interface*)

   Komponen ini merupakan kumpulan *library* yang dapat digunakan untuk menjalankan dan mengembangkan program Java pada *handheld devices*.

c. *Tools* lain untuk pengembangan aplikasi Java semacam emulator

Selain itu, J2ME juga dibagi menjadi 2 buah bagian yang dikenal dengan istilah *configuration* dan *profile*. Penjelasannya adalah sebagai berikut :

a. J2ME *Configuration*

   J2ME *Configuration* mendefinisikan lingkungan kerja J2ME *runtime.* Oleh karena setiap *handheld devices* memiliki fitur yang berbeda-beda, maka J2ME *configuration* dirancang untuk menyediakan *library standard* yang mengimplementasikan fitur *standard* dari sebuah *handheld services*.

   Terdapat 2 kategori J2ME Configuration yaitu:

   - CLDC (*Connected Limited Device Configuration*)

     Kategori ini umumnya telah digunakan untuk aplikasi Java pada perangkat dengan memori 160 – 512 KB seperti *handphone* dan PDA.

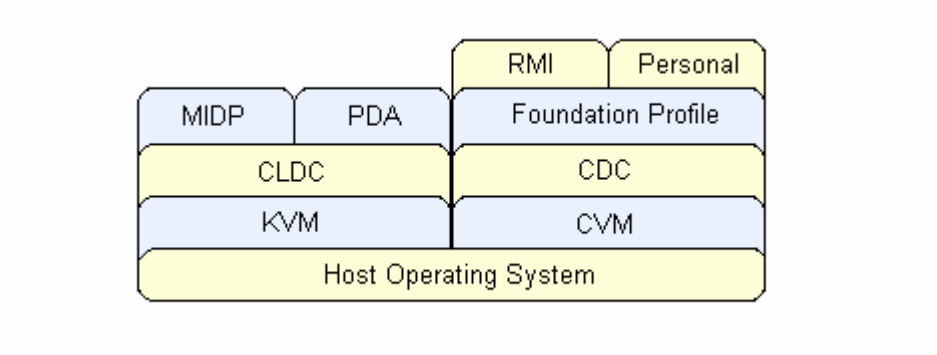
   - CDC (*Connected Device Configuration*)

     Umumnya digunakan pada perangkat dengan memori setidaknya 2 MB seperti internet TV, Nokia *Communication* dan *CarTelevision*. (Wicaksono, 2002, p. 5-9)

b. J2ME Profile

Jika J2ME *configuration* digunakan untuk menyediakan *library* Java untuk implementasi fitur-fitur standar pada sebuah *handheld devices*, maka J2ME Profile menyediakan implementasi tambahan yang sangat spesifik bagi *handheld devices* tertentu. Sebuah *handphone* pasti memiliki kemampuan standar yaitu dapat digunakan untuk komunikasi suara. Namun terdapat perbedaan spesifikasi misalnya *handphone* yang satu dapat memutar *file audio* MP3, sementara yang lain tidak. J2ME Profile akan memberikan infomasi tentang kemampuan device.

Terdapat lima kategori J2ME *Profile* yaitu *Mobile Interface Device Programming* (MIDP), *Foundation Profile* (FP), *Personal Profile, RMI Profile,* dan *Personal Digital Assistance Profile*.




Gambar 2.2. Arsitektur Java 2 Micro Edition

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.10

Secara umum J2ME *Profile* dan *Configuration* yang membentuk Arsitektur Java 2 Micro Edition dapat digambarkan pada Gambar 2.1. Pada pembuatan aplikasi *game multiplayer* ini yang digunakan adalah J2ME MIDP Profile, khusus MIDP versi 1.0 dimana menyediakan *library-library* Java untuk implementasi dasar antar muka atau *Graphical User Interface* (GUI), implementasi jaringan atau *networking*, *database* dan timer. Sedangkan untuk J2ME Configuration, menggunakan CLDC karena implementasinya pada peralatan (*handphone*) dengan *memory* antara 128 hingga 512 *Kilobytes*.

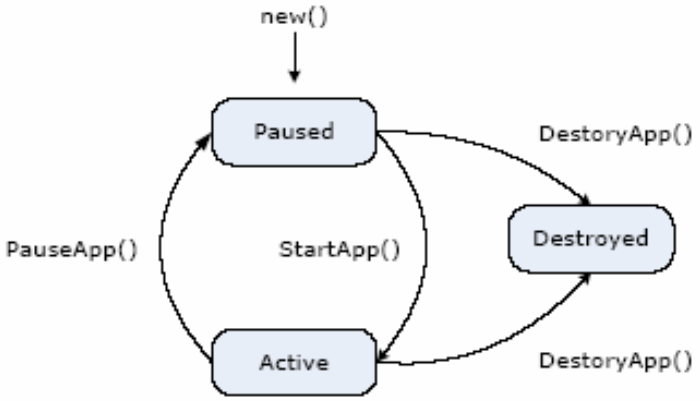
2.2.2. MIDlet pada J2ME

MIDlet merupakan sebuah aplikasi yang dibuat menggunakan J2ME dengan profile MIDP. Seperti sebelumnya telah dijelaskan, MIDP dikhususkan untuk digunakan pada perangkat yang menggunakan kemampuan CPU, *memory*, keyboard dan layar yang terbatas.

Hal awal yang harus diketahui dalam membuat suatu aplikasi MIDlet yaitu menyangkut Siklus Hidup atau *Lifecycle*. *Lifecycle* dari sebuah MIDlet ditangani oleh Application Management Software (AMS) dimana merupakan lingkungan tempat siklus sebuah MIDlet mulai dari diciptakan, dijalankan, dihentikan hingga dihapuskan. AMS sering disebut pula sebagai Java Application Manager (JAM). (Hartanto, 2002, p. 14)

Kelas MIDlet sendiri merupakan paket dari javax.microedition.midlet yang nantinya mendefinisikan MIDP dan interaksi dengan perangkat *handheld* bersangkutan. Kelas ini merupakan kelas utama dalam sebuah MIDlet dimana semua aplikasi merupakan turunan dari kelas ini.

MIDlet memiliki beberapa state atau kondisi yang berbeda-beda, yaitu `actived()`, `destroyed()` dan `paused()`. MIDlet dapat dianalogikan sebagai sebuah mobil dimana ada kondisi starter mesin, mobil berjalan, mobil berhenti dan mesin dimatikan. Agar lebih jelas, *Lifecycle* MIDlet digambarkan pada gambar 2.3.




Gambar 2.3. *Lifecycle* dari sebuah MIDlet

Dari gambar 2.3. dapat dijelaskan bahwa MIDlet memiliki 3 kondisi yaitu :

1) *Paused*

   Terjadi saat MIDlet pertama kali diciptakan dan selesai diinisialisasi dan tidak melakukan aksi apapun. Secara garis besar, status MIDlet p*aused* adalah pada saat :

   - Setelah MIDlet dibuat menggunakan konstruktor `new()`.
   - Dari status *active*, dilakukan pemanggilan MIDlet dengan fungsi `MIDlet.pauseApp()` atau fungsi `MIDlet.notifyPaused()`.
   - Dari status *active*, namun ketika start() terjadi kesalahan berupa exception: `MIDletStateChangeException`.

   Pada saat staus ini terjadi, MIDlet tidak boleh sedang mengunci resource, misalnya melakukan penguncian sebuah file sehingga proses lain tidak bisa melakukan penulisan ke dalam file tersebut

2) *Active*

   Status ini terjadi saat MIDlet aktif/berjalan normal setelah dilakukan pemanggilan terhadap fungsi `MIDlet.startApp()`.

3) *Destroyed*

   Status ini terjadi saat MIDlet berhenti dijalankan yang biasanya identik dengan *exit program*, melalui pemanggilan fungsi `MIDlet.destroyApp()` atau `MIDlet.notifyDestroyed()`, semua resource yang sebelumnya digunakan oleh MIDlet termasuk memori, akan dibersihkan. (Wicaksono, 2002, p. 45-46)

Adapun urutan eksekusi sebuah MIDlet adalah sebagai berikut :

1. AMS (*Application management Software*) menginisialisai *object* MIDlet sehingga MIDlet berada dalam status *Paused*.
2. Saat AMS memutuskan bahwa MIDlet sudah saatnya dijalankan, maka AMS akan melakukan pemanggilan fungsi `MIDlet.startApp()`.
3. Saat AMS memutuskan bahwa MIDlet harus dinonaktifkan sementara, maka AMS akan melakukan pemanggilan fungsi `MIDlet.pauseApp()`.
4. Saat AMS memutuskan bahwa MIDlet harus dinonaktifkan atau dihentikan, maka AMS akan memanggilan fungsi `MIDlet.destroyApp()` dan pembebasan *resource* yang sebelumnya digunakan akan dilakukan.

2.2.3. Contoh Aplikasi MIDlet

Agar lebih jelas, berikut ini akan diberikan satu contoh aplikasi yang memperlihatkan *Lifecycle* dari pemrograman pada MIDlet. Aplikasi yang dibuat akan menampilkan kata *"Hello world"*. Aplikasi dieksekusi dan dijalankan di emulator J2ME yaitu J2MEWTK (Java 2 *Micro Edition Wireless Toolkit*) pada komputer platform Windows. Kode programnya sebagai berikut :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello
extends MIDlet implements CommandListener{

private Form mainForm;

public Hello () {
      mainForm = new Form("Hai!");
      mainForm.append("Hello world");
      mainForm.addCommand(new Command
      ("exit",Command.EXIT,0));
      mainForm.setCommandListener(this); }

public void startApp(){
      Display.getDisplay(this).setCurrent(mainForm); }

public void pauseApp() {}

public void destroyApp(boolean unconditional) {}

public void commandAction (Command c, Displayable d) {
      notifyDestroyed(); } }
```

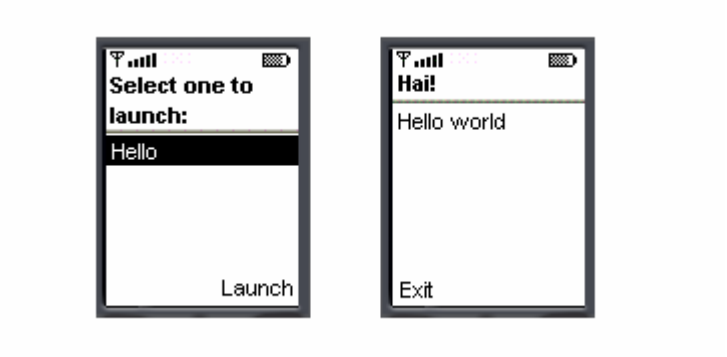
Bila diperhatikan kode program di atas, aplikasi MIDlet yang dibuat merupakan turunan dari kelas MIDlet., yang dituliskan sebagai berikut :

```
public class Hello extends MIDlet implements CommandListener
```

Kelas `Hello` merupakan kelas utama sebuah aplikasi MIDlet dimana merupakan turunan dari kelas MIDlet dan semua kelas yang ada harus merupakan turunan kelas MIDlet. Kelas MIDlet merupakan kelas abstrak dimana harus terdapat ketiga fungsi yaitu :

- ```
  public void startApp(){
        Display.getDisplay(this).setCurrent(mainForm); }
  ```
- ```
  public void pauseApp() {}
  ```
- ```
  public void destroyApp(boolean unconditional) {}
  ```
  ```

Hasil eksekusi program di atas ditampilkan pada Gambar 2.4 sebagai berikut :




Gambar 2.4. Contoh Tampilan MIDlet

Setelah diinisialisasi, kondisi awal MIDlet berada pada keadaan *"pause"* yang menunjuk `pauseApp()`. Kondisi ini terjadi secara otomatis sesaat setelah MIDlet diinisialisasi, ditunjukkan gambar 2.4. sebelah kiri. Saat tombol (*soft button*) *handheld* yang mengacu ke perintah *launch* ditekan, MIDlet segera berada pada keadaan *"active"* dengan melakukan perintah yang ada pada `startApp()` yaitu menampilkan Form `mainForm` ke layar. Hal ini terjadi hingga adanya penekanan tombol *exit* yang akan mengubah state MIDlet menjadi *"destroy"* setelah pemanggilan fungsi standar `destroyApp()` dilaksanakan. Proses ini sekaligus melaksanakan proses *cleanup* terhadap *garbage collector* yang sebelumnya digunakan selama aplikasi berjalan.

## 2.3. Pemrograman GUI pada MIDlet

Selain keterbatasan media penyimpanan dan alokasi *memory*, keterbatasan perangkat *handheld* biasanya berupa keterbatasam layar atau *screen* dan masukan input. Keterbatasan tersebut menyebabkan perlunya teknik pemrograman yang berbeda dengan pemrograman pada aplikasi untuk komputer.

Pemrograman pada MIDlet ditekankan pada pemrograman antar muka secara grafik atau *Graphical User Interface* (GUI). CLDC tidak menyediakan fungsi-fungsi untuk GUI, namun fungsi ini akan ditangani oleh MIDP. Semua fungsi antar muka berbasis window ditangani oleh paket javax.microedition.lcdui. *User Interface* dari MIDP terdiri dari API-API (*Application Programming Interface*) *Low Level* dan *High Level*. (Hartanto, 2002, p.15).

Pada bagian berikut akan dijelaskan beberapa pemrograman J2ME menggunakan MIDlet yang banyak digunakan untuk pembuatan aplikasi *game multiplayer* ini.

### 2.3.1. *High Level* API

Pemrograman aplikasi menggunakan *High Level* API, lebih dianjurkan karena aplikasi yang dibuat untuk fungsionalitas interaksi *user* dengan *handheld* lebih *port*able. Dengan banyak perangkat *handheld* yang berbeda, bila aplikasi ditampilkan di *handheld* lain, tampilannya akan sama. Pada pemrograman *High Level*, kelas-kelas untuk manajemen pemrograman diturunkan dari kelas *Screen* (`javax.microedition.lcdui.Screen`).

#### 2.3.1.1. *Display*

*Display* merupakan kelas yang menyediakan fungsi-fungsi untuk manajemen layar, menampilkan objek *screen* dan menyediakan informasi tentang properti dari perangkat *handheld* yang digunakan. Akses ke layar didapatkan dengan fungsi statik `getDisplay()` pada kelas *Display*. Penggunaan fungsi ini umumnya dilakukan pada fungsi `startApp()` untuk menjadikan status MIDlet menjadi "*active*".

Setelah pemanggilan fungsi `getDisplay()`, kita dapat menentukan oblek mana yang ditampilkan menggunakan fungsi `setCurrent()`. Contohnya dapat dilihat pada contoh pembuatan aplikasi MIDlet pada sub bab 2.2.3.

#### 2.3.1.2. *Screen*

Seperti yang telah disebutkan sebelumnya, objek *screen* menyediakan fungsionalitas untuk interaksi antara *user* dengan *handheld*. Objek ini memiliki empat jenis turunan yang umum digunakan, yaitu :

- *TextBox*

  *Object TextBox* menyediakan media untuk menerima masukkan *string* melalui tombol *keypad handheld* oleh pengguna, misalkan masukan nama, *password* dan informasi lain yang berbasiskan teks. Kelas *TextBox* merupakan

turunan dari `javax.microedition.lcdui.screen`. Konstruktor kelas *TextBox* ini adalah :

```
Public TextBox
(String title,String Text,int maxSize,int constraints)
```

Adapun Objek *TextBox* tidak akan dijelaskan secara jelas karena objek ini tidak digunakan pada pembuatan aplikasi game multiplayer.

- *Alert*

Kelas *Alert* memiliki fungsi untuk menyampaikan informasi ke *user* yang ditampilkan dalam jangka waktu relatif singkat (dalam waktu tertentu) sebelum pindah ke objek *screen* lainnya. Pada umumnya digunakan saat menyampaikan peringatan, terjadi *error*, atau informasi konfirmasi singkat ke pengguna. *Alert* juga merupakan kelas turunan dari `javax.microedition.lcdui.screen`. Konstruktor kelas *Alert* ini ada dua yaitu :

```
public Alert (String title);

public Alert (String title, String alertText, Image
      alertImage, AlertType type);
```

Parameter yang digunakan pada pambutan aplikasi ini antara lain :

a. `String title`

Digunakan untuk memberikan judul atau *title* pada layar atau dibiarkan kosong (*null)* sehingga tidak ada judul pada layar yang ditampilkan.

b. `String alertText`

Digunakan untuk memberikan pesan berupa teks pada layar *Alert* atau dibiarkan *null* sehingga tidak ada pesan yang ditampilkan.

c. `AlertType type`

Parameter ini dapat diisi antara lain :

- `AlertType.ALARM`
- `AlertType.INFO`
- `AlertType.WARNING`
- `AlertType.CONFIRMATION`
- `AlertType.ERROR`

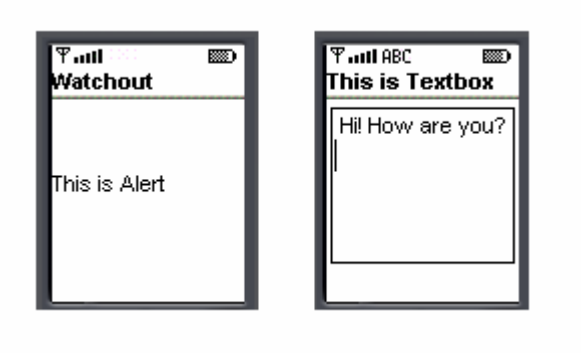
Perbedaan masing-masing tipe terutama pada efek suara (*sound*) yang diperdengarkan oleh *handphone*.

Contoh program yang menggunakan *Alert* :

```
private Display display;
public AlertDemo(){}
public void startApp() {
      display = Display.getDisplay(this);
      TextBox tbox = new TextBox("This is Textbox ",
      " Hi! How are you? ",256,TextField.ANY);
      Alert tmAlert = new Alert("Watchout","This is Alert"
            ,null,AlertType.WARNING);
      Display display = Display.getDisplay(this);
      display.setCurrent(tmAlert, tbox);
      }

public void pauseApp() {// Do Nothing}

public void destroyApp(boolean unconditional){// Do Nothing}
      }
```




Gambar 2.5. Contoh tampilan *Alert*

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.66

Pada contoh program diatas, *Alert* akan ditampilkan sebelum TextBox dimunculkan di layar. *Alert* akan tampil kurang lebih sekitar dua detik dan secara otomatis layar akan berpindah ke TextBox, seperti ditampilkan pada gambar 2.6.

- *List*

Kelas *List* atau `javax.microedition.lcdui.List` merupakan kelas turunan dari kelas abstrak `javax.microedition.lcdui.screen` yang digunakan untuk menyediakan masukan berupa pilihan ganda (*multiple choice*). Konstruktor dari kelas List ini ada dua yaitu :

```
public List(String title, int listType)

public List(String title,
     int listType String[] listElement, Image[] listImages)
```

Parameter-parameter yang digunakan :

*a.* `String title`

Memberikan *title* pada saat *List* ditampilkan.

*b.* `Int list type`

Untuk mendefinisikan tipe *list* yang digunakan, diantaranya dapat berupa :

- `Choice.EXCLUSIVE`

  Setiap satu list, hanya dapat memilih satu pilihan saja

- `Choice.IMPLICIT`

  Sama halnya dengan `Choice.EXCLUSIVE`, hanya tampilannya saja yang berbeda.

- `Choice.MULTIPLE`

  Digunakan untuk bisa memilih lebih dari satu pilihan pada satu *list*.

*c.* `String[] list Element`

Merupakan *array* dari element *list* yang akan ditampilkan

*d.* `Image[] listImage`

Merupakan *array* dari gambar `(javax.microedition.lcdui.Image)` element *list* yang ditampilkan.

Jika *konstruktor* yang digunakan adalah jenis *konstruktor* pertama (`public List(String title, int listType)`), maka akan dibuat sebuah *list* kosong yang kemudian di dalamnya dapat ditambahkan, disisipkan atau dilakukan penggantian pilihan dalam *list*. Fungsi yang berkaitan dengan hal ini adalah :

1) `public int append (String element,Image image)`

   Digunakan untuk menambahkan satu elemen pada daftar pilihan dengan gambar tertentu. Jika tidak menampilkan gambar, maka parameter *Image* harus di-*set null*.

2) `public int insert (int index, String element,Image image)`

   Digunakan untuk menyisipkan satu elemen pada daftar pilihan di lokasi yang ditentukan oleh nilai parameter *index*. Dengan gambar tertentu. Jika tidak menampilkan gambar,parameter *Image*s harus di-*set null*.

Contoh *source code* penggunaan *List*:

```
private Display display;
public listDemo(){}
```

```
public void startApp() {
      display = Display.getDisplay(this);
      List myList = new List("This isList",Choice.IMPLICIT);
      myList.append("Choice 1",null);
      myList.append("Choice 2",null);
      myList.append("Choice 3",null);
      myList.insert(1,"Inserted Choice",null);
      display.setCurrent(myList); }

public void pauseApp() { }

public void destroyApp(boolean unconditional){} }
```

Pada contoh program diatas, sebuah *List* akan menampilkan 4 macam pilihan dimana salah satunya menggunakan parameter *insert*. Dengan mengganti `Choice.IMPLICIT` pada *source code* menjadi `Choice.EXCLUSIVE` atau `Choice.MULTIPLE`, akan menghasilkan tipe dan tampilan *List* yang berbeda, seperti pada gambar 2.7. berikut ini.


Gambar 2.6. Contoh Tampilan *List*

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.71

- *Form*

  Bekerja dengan *Form* memungkinkan adanya tampilan beberapa komponen *Graphical User Interface* seperti daftar pilihan (*list*) atau masukan teks (*textfield*) dalam satu layar. *Form* diimplementasikan oleh kelas `javax.microedition.lcdui.Form`. *Form* dapat menampung komponen-komponen yang disebut *item* dalam satu layar, yaitu *textfield, image, datefield, gauge, choice groups* dan *list*. Ada dua konstruktor *form* yang umum digunakan yaitu :

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

Konstruktor pertama secara sederhana menyediakan sebuah *form* dengan judul atau *title* yang diberikan saja. Sedangkan konstruktor yang kedua, selain mendefinisikan *title* juga mendefinisikan *item-item* apa saja yang akan ada dalam *form* tersebut. Pada saat *item-item* tertentu ditambahkan pada *form*, maka secara otomatis *item-item* tersebut akan ditambahkan secara teratur dari atas ke bawah. Perhatikan gambar 2.8. yang memperlihatkan Form yang kosong.

Selain *item* juga terdapat beberapa fungsi penting yang dapat ditambahkan pada *form* sebagian di antaranya adalah :

a. `Public int append(Image img)`

   Digunakan untuk menambahkan gambar ke dalam *form*. *Item* ini diimplemetasikan dalam kelas `javax.microedition.Image`.

b. `Public int append(String str)`

   Digunakan untuk menambahkan *string* ke dalam *form*.

c. `Public int append(Item itm)`

   Digunakan untuk menambahkan *item* ke dalam *form.*


Gambar 2.7. Contoh *Form*

Komponen-komponen yang dapat diletakkan dalam sebuah *form* kosong merupakan komponen-komponen yang memiliki kelas yang merupakan kelas turunan dari kelas *abstrak* yaitu `javax.microedition.lcdui.Item`.. Komponen-komponen tersebut antara lain :

- *Choice Group*

  Kelas ini menyediakan komponen yang mirip dengan *list* yakni untuk menyediakan daftar pilihan

- *Datefield*

  Kelas ini menyediakan komponen untuk memasukkan informasi tanggal dan waktu.

- *Gauge*

  Kelas ini menyediakan komponen grafik horizontal yang biasa dipakai untuk memberikan gambaran persentase suatu proses yang sedang berjalan, misalnya saat proses *downloading*.

- *Image* dan *ImageItem*

  Kelas ini menyediakan komponen grafik untuk manipulasi gambar.

- *StringItem*

  Kelas ini tampilannya mirip dengan *TextBox* namun komponen teks *string* yang ditampilkan, tidak bisa diedit oleh *user*.

- *TextField*

  Kelas ini tampilan dan fungsinya mirip dengan *TextBox* dimana menyediakan komponen untuk masukkan teks *string* dari *user*. Beberapa *item* yang dipakai sehubungan dengan lingkup tugas akhir ini akan dibahas secara mendalam sedangkan *item-item* lain yang tidak berhubungan tidak akan dibahas.

  Item yang banyak digunakan pada tugas akhir ini antara lain :

A. *StringItem*

*Object StringItem* dipakai untuk meletakkan *object* teks *string* pada *form* yang tidak dapat diubah oleh *user*. *StringItem* diimplementasikan oleh kelas `javaxmicroedition.lcdui.StringItem` yang merupakan kelas turunan dari kelas abstrak `javax.microedition.lcdui.Item`. Konstruktor dari kelas *StringItem* ini adalah :

```
public StringItem (String label,String Content)
```

Parameter-parameter ini memiliki fungsi sebagai berikut:

- `String label` digunakan untuk memberikan *title* pada *object* string yang ditampilkan.
- `String content` merupakan isi dari teks yang akan ditampilkan pada *form*.

Contoh penggunaannya sebagai berikut :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class stringItemDemo extends MIDlet{
private Display display;

public stringItemDemo(){ }

public void startApp() {
      display = Display.getDisplay(this);
      Form f = new Form("Greeting");
      StringItem si =
            new StringItem(" I say:\t","Hi Guys");
      f.append(si);
      display.setCurrent(f); }

public void pauseApp() { // Do Nothing }

public void destroyApp(boolean unconditional)
      { //Do Nothing }}
```

Bila *source code* di atas dijalankan, tampilannya tampak seperti gambar 2.9.


Gambar 2.8. Contoh tampilan *StringItem*

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.89

Fungsi-fungsi yang ada dalam kelas ini adalah :

- `public void setText(String teks)`

  Menetapkan teks sesuai parameter teks yang diberikan pada *object StringItem*.

- `public void setLabel(String lbl)`

  Menetapkan *label* pada *object StringItem* sesuai parameter `lbl` yang diberikan pada *object StringItem*.

- `public String getText()`

  Menghasilkan teks *string* yang ada pada *object StringItem*.

- `public String getLabel()`

  Menghasilkan *label* yang ada pada *object StringItem*.

B. *TextField*

*Object TextField* digunakan untuk memasukkan objek *string* yang dapat diubah secara langsung pada *form*. *Object* ini diimplemetasikan oleh kelas `javax.microedition.lcdui.TextField` yang merupakan kelas turunan dari kelas `abstrak javax.microedition.lcdui.Item`. Objek *TextField* dapat dikatakan sama dengan *TextBox* yang telah dibahas sebelumnya, dimana perintah yang digunakan keduanya hanya berbeda sedikit. Konstruktor dari kelas *TextField* ini adalah :

```
public TextField(String title,String Text,
      int maxSize,int constraints)
```

Parameter-parameter yang digunakan antara lain :

- `String Title`

  Digunakan untuk memberikan *title* pada *TextField* yang aktif.

- `String Text`

  Digunakan untuk memberikan nilai awal (kata yang langsung muncul) ketika *TextField* yang dimaksud ditampilkan.

- `int maxSize`

  Digunakan untuk memberikan batasan jumlah karakter yang dapat dimasukkan pada sehingga *user* tidak akan dapat memasukkan karakter yang jumlahnya melebihi batas maksimal *TextField*.

- `int constraint`

  Beberapa nilai yang digunakan dalam parameter ini yaitu :

  a. `TextField.ANY`

     Sembarang karakter akan dapat dimasukkan dalam *TextField*

  b. `TextField.EMAILADDR`

     Hanya teks berupa alamat *email* saja yang dapat dimasukkan dalam *TextField* ini.

c. `TextField.NUMERIC`

Hanya teks berupa angka saja yang dapat dimasukkan dalam *TextField* ini.

d. `TextField.PHONENUMBER`

Hanya teks berupa nomor telepon saja yang dapat dimasukkan dalam *TextField* ini.

e. `TextField.URL`

Hanya teks berupa alamat *web* atau URL saja yang dapat dimasukkan dalam *TextField* ini.

f. `TextField.PASSWORD`

Semua teks *string* yang dimasukkan akan ditampilkan dalam karakter ' * ' seperti saat memasukkan *password*.

Beberapa fungsi penting dalam kelasTextField ini antara lain :

- `public String getString()`

  Menghasilkan *string* yang saat ini ada pada *textField*

- `public void setString(String str)`

  Menetapkan *string* pada *TextField* menjadi '`str`' pada parameter fungsi diatas

- `public void insert(String str,int position)`

  Menyisipkan *string* 'str' pada parameter fungsi di atas pada *TextField* mulai dari *position*.

Fungsi yang terdapat pada *TextField* , hampir semuanya terdapat pada fungsi yang digunakan *TextBox*. Bedanya, *TextField* harus berada di dalam *form*, sedangkan *TextBox* tidak perlu. Berikut ini diberikan contoh penggunaan *TextField* yang sedikit berbeda dengan penggunaan *TextBox* :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class textFieldDemo extends MIDlet{
private Display display;

public textFieldDemo (){}

public void startApp() {
      display = Display.getDisplay(this);
      TextField lname = new TextField
      ("Login Name","madeyudistira",25,TextField.ANY);
      TextField pval = new TextField
```

```
        ("Password","secret",45,TextField.ANY|TextField.
        PASSWORD);
        Form f = new Form("Sign in");
        f.append(lname);
        f.append(pval);
        display.setCurrent(f); }

    public void pauseApp() {// Do Nothing }

    public void destroyApp(boolean unconditional)
        {// Do Nothing } }
```

Program di atas menampilkan dua *TextField*, dimana *field* nama menggunakan `TextField.ANY` dan *field password* menggunakan `TextField.PASSWORD` seperti yang tampak pada Gambar 2.9.


Gambar 2.9. Contoh Tampilan *TextField.*

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.95

2.3.2. *Low Level* API

Pada pemrograman *High Level*, tampilan antarmuka dibuat standar dimana ada masukan untuk teks, *password*, tampilan pesan dan sebagainya. Namun ada komponen-komponen yang tidak disediakan dalam hal-hal lain seperti untuk keperluan pembuatan *game*, seperti fungsi-fungsi untuk mengolah gambar atau yang *level*nya sudah sampai pada pengaturan *pixel*. Pada pemrograman *Low level*, kita akan mendapatkan fungsionalitas yan lebih spesifik ke jenis *hendheld* yang digunakan, namun belum tentu bisa dijalankan di *handheld* lain. *Low Level* API berbasis pada *class Canvas* (`javax.microedition.lcdui.Canvas`). Berikut akan dibahas pemrograman *low level* yang digunakan untuk membuat *game* ini.

2.3.2.1. *Canvas* dan *Graphics*

Kelas *Canvas* merupakan kelas abstrak sekaligus turunan dari kelas *Displayable.* Penggunaan *Canvas* biasanya bersamaan dengan kelas *Graphics*,

karena kelas *Graphics* yang menyediakan objek-objek grafik dua dimensi dan fungsi untuk manipulasi grafik *level* rendah.

Contoh penggunaan kelas *Canvas* :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class CanvasDemo extends MIDlet {
      private Display display;
      public CanvasDemo () { }

public void startApp() {
      display = Display.getDisplay(this);
      Canvas Canvas = new myCanvas();
      display.setCurrent(Canvas); }

public void pauseApp() {// Do Nothing }

public void destroyApp(boolean unconditional)
      {// Do Nothing }

public class myCanvas extends Canvas{
      public void paint(Graphics g){
            g.setColor(0,0,255);
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(255,255,255);
            g.drawString(
                  "Hello guys!!",20,20,g.TOP|g.LEFT);
            } } }
```

Tampak bahwa kelas `myCanvas` merupakan turunan dari kelas *Canvas*, dimana melakukan pemanggilan fungsi `public void paint(Graphics g)` yang nantinya melakukan pencetakan gambar dengan objek *Graphics* sebagai parameternya.

Fungsi penting yang berkaitan dengan kelas *Canvas* dan *Graphics*:

- `public int getWidth()`

  Digunakan untuk menghasilkan nilai *integer* yang menunjukkan nilai lebar atau absis maksimal layar *handheld*.

- `public int getHeight()`

  Digunakan untuk menghasilkan nilai *integer* yang menunjukkan nilai tinggi atau ordinat maksimal layar.

- `public void setColor(int red,int green,int blue)`

  Digunakan untuk menetapkan suatu warna pada kelas *Graphics* dimana merepresentasi pencampuran warna dasar *Red, Green, Blue*.

Informasi dukungan warna pada *handheld* dapat diperoleh dengan menggunakan fungsi `isColor()` pada kelas *Display*.

- `public void fillRect(int x,int y,int width, int height)`
  Digunakan saat menggambar segiempat dengan koordinat kiri atas (x,y) sampai batas lebar (*width*) dan tinggi (*height*) dengan warna yang sedang digunakan.
- `public void drawString(String str, int x, int y, int anchor)`
  Digunakan untuk menambahkan sebuah *string* pada posisi x,y dengan aturan sesuat nilai *anchor*. Nilai anchor merupakan konstanta pada kelas *Graphics* yang menentukan peletakkan teks, dapat bernilai `TOP` untuk rata atas, `LEFT` untuk rata kiri, `RIGHT` untuk rata kanan, dan `BASELINE` untuk rata tengah.

Bila ke contoh di atas dijalankan maka hasilnya adalah seperti pada gambar 2.10.


Gambar 2.10. Contoh Tampilan *Canvas* dan *Graphics*

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.104

2.3.2.2. *Font*

Kelas *Font* yang ditangani oleh `javax.microedition.lcdui`, digunakan untuk memanipulasi tipe font yang disediakan *handheld*. Font tidak dibuat oleh aplikasi namun hanya merubah tampilan *font* yang diinginkan menurut tipe, ukuran dan atribut lain unutk ditampilkan di *handheld*. Ada hal yang perlu diketahui saat menggunakan kelas Font ini, yaitu :

- 3 macam ukuran font :
  a. `Font.SIZE_SMALL` untuk ukuran kecil

b. `Font.SIZE_MEDIUM` untuk ukuran sedang

c. `Font.SIZE_LARGE` untuk ukuran besar.

- 4 macam corak huruf :

a. `Font.STYLE_PLAIN` untuk corak huruf standar

b. `Font.STYLE_ITALIC` untuk corak huruf miring

c. `Font.STYLE_BOLD` untuk corak huruf tebal

d. `Font.STYLE_UNDERLINED` untuk corak huruf bergaris bawah

Tidak ada konstruktor untuk kelas ini, umumnya cara menggunakannya dengan memanggil fungsi statik `getFont()` seperti berikut:

```
Font f = Font.getFont
      (FACE_SYSTEM, STYLE_PLAIN, SIZE_SMALL);
```

### 2.4. Manajemen *Event*

Ketika terjadi interaksi antara *user* dengan perangkat *handheld*, maka akan dihasilkan suatu event. Saat penekanan tombol, memasukkan teks, ataupun memilih menu, sistem akan memproduksi suatu *event* yang nantinya memberi tahu bahwa telah terjadi suatu event yang kemudian ditindaklajuti aplikasi dengan melakukan suatu *action* yang dikehendaki *user*.

Pada pembuatan tugas ini, semua menggunakan penanganan event tingkat tinggi dengan mengimplementasikan *CommandListener*. Pada contoh Aplikasi MIDlet pada sub bab 2.2.2, telah diperlihatkan juga contoh penanganan *event* sederhana menggunakan *CommandListener* yang digunakan saat *exit* aplikasi.

Kelas *command* (`javax.microedition.lcdui.Command`) merupakan kelas yang membungkus informasi sebuah aksi. Dari informasi yang dibungkus oleh kelas ini, maka aplikasi bisa menentukan aksi apa yang akan dilakukan. Fungsi yang berkaitan dengan manajemen *event* objek *Command* ini adalah :

a. `public void addCommand(Command cmd)`

   Mengasosiasikan suatu sebuah objek GUI yang bersangkutan dengan objek `Command cmd`.

b. `Public void removeCommand(Command cmd)`

   Menghapuskan asosiasi objek GUI yang bersangkutan dengan objek *Command cmd* yang sebelumnya dibuat.

c. `Public void setCommandListener(CommandListener l)`

Mengasosiasikan objek GUI yang bersangkutan dengan *interface CommandListenerl*.

Ketiga fungsi tersebut merupakan fungsi yang diturunkan oleh kelas *Displayable.* Konstruktor kelas *Command* sendiri adalah :

```
public Command(String label, int commandType, int prio)
```

Parameter yang digunakan dalam konstruktor kelas *command* di atas adalah :

1) `String label`

Untuk memberikan *label* pada perintah yang ada

2) `Command type`

Merupakan tipe perintah yang ada. beberapa nilai yang diperbolehkan yaitu :

- `Command.BACK`

  Fungsi ini digunakan untuk mengasosiasikan perintah untuk kembali ke tampilan sebelumnya (seperti halnya perintah *back* pada *browser internet*).

- `Command.CANCEL`

  Fungsi ini digunakan untuk mengasosiasikan perintah untuk pembatalan pengaksesan tampilan dan sebagai respons akan ditampilkan tampilan sebelumnya. Secara respons, hal ini sama dengan `Command.BACK`.

- `Command.OK`

  Fungsi ini digunakan untuk mengacu kepada jawaban positif atas suatu hal, misalnya saja melakukan suatu proses tertentu.

- `Command.HELP`

  Fungsi ini digunakan untuk menampilkan fasilitas *help* pada layar.

- `Command.STOP`

  Fungsi ini digunakan untuk menghentikan suatu proses yang sedang berlangsung.

- `Command.EXIT`

  Fungsi ini digunakan untuk keluar dari suatu aplikasi.

3) `int prio`

   Merupakan suatu nilai *integer* yang menggambarkan prioritas dari perintah-perintah yang ada. Nilai prioritas ini diisi dengan bilangan positif (dimulai dari angka 1), dimana nilai terkecil merupakan nilai prioritas tertinggi. Nilai prioritas ini juga akan mempengaruhi penempatan suatu perintah pada layar *handheld*.

Reaksi yang diberikan oleh aplikasi sebagai aksi pada objek *Command*, hanya bisa dilakukan dengan mengimplementasikan *interface* `CommandListener` dan objek yang akan diasosiasikan dengan *interface* tersebut harus menggunakan fungsi `setCommandListener()`. Agar lebih jelas perhatikan contoh penggunaan CommandListener berikut:

```
public class TextBoxWithCmd extends MIDlet
      implements CommandListener{

private Display display;
private Command exitCmd =
      new Command("Keluar",Command.EXIT,1);

public TextBoxWithCmd(){}

public void startApp(){
      display = Display.getDisplay(this);
      TextBox t = new TextBox("This is TextBox ","",256,
            TextField.ANY);
      Integer maxSize = new Integer(t.getMaxSize());
      String toPrint = "Jumlah maksimal karakter ";
      t.setString(toPrint + " = " + maxSize.toString());
      t.addCommand(exitCmd);
      t.setCommandListener(this);
      display.setCurrent(t); }

public void() { this.destroyApp(true); }

public void pauseApp() { // Do Nothing }

public void destroyApp(boolean unconditional)
      { // Do Nothing }

public void commandAction(Command c, Diplayable d)
      { String lbl = c.getlabel();
      if (lbl.equals("Keluar")) {
            keluarProg();  } }

      }
```

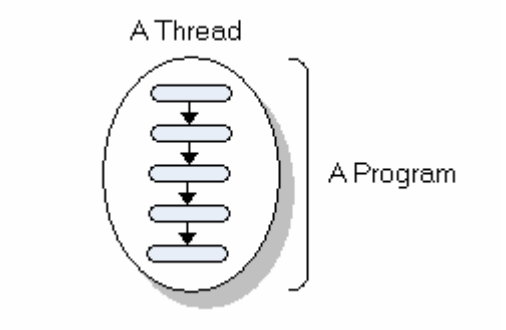
Bila kode program di atas dijalankan, tampilannya akan seperti Gambar 2.11. Program yang digunakan adalah program contoh tampilan *TextBox* dengan penambahan *ComandAction* "Keluar" untuk keluar dari aplikasi.


Gambar 2.11. Contoh *TextBox* dengan *Command*

Sumber : Ady Wicaksono. *Pemrograman Aplikasi Wireless dengan Java,* (Jakarta: Elex Media Komputindo, 2002), p.137

## 2.5. Pembuatan *Thread*

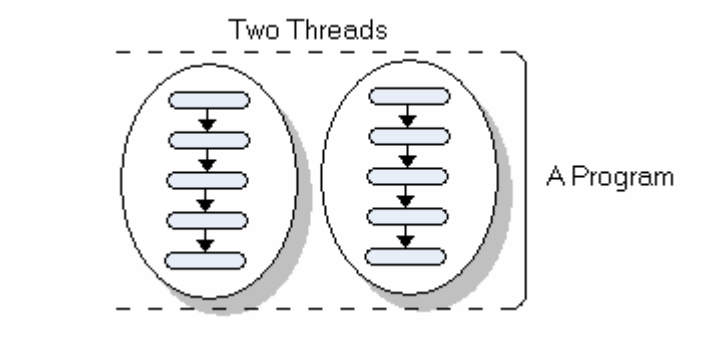
Bila kita membuat suatu program sederhana, biasanya proses yang terjadi adalah berurutan, maksudnya suatu proses akan berjalan bila suatu proses lain telah selesai dijalankan, dianalogikan seperti pada Gambar 2.12.




Gambar 2.12. Proses pada program satu *thread*

Sumber : Ady Wicaksono. *Pemrograman Internet dan XML pada Ponsel dengan MIDlet Java,* (Jakarta: Elex Media Komputindo, 2002), p.82

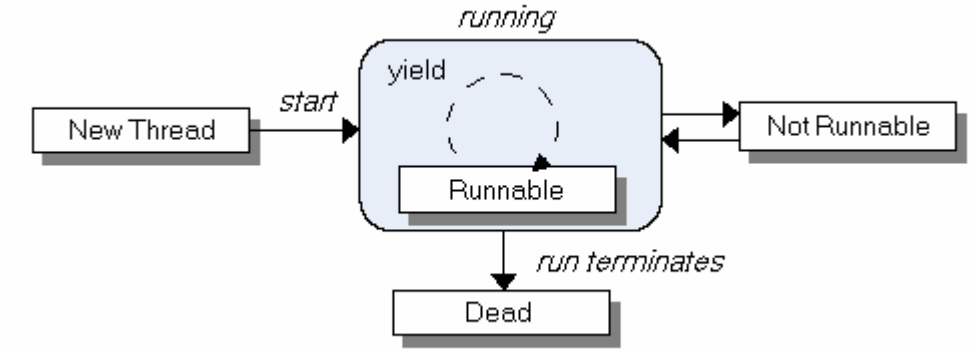
Ide tentang *thread* relatif sederhana, yaitu melakukan lebih dari satu proses dalam satu waktu. Jadi dua atau lebih proses dapat berjalan seolah-olah bersama-sama dimana masing-masing tidak menggangu satu sama lain.Bila kita menggunakan browser dimana saat proses sedang *downloading* data, *user* dapat melakukan scrolling page, menghentikan transfer data, atau menutup browser tersebut, seperti itulah *thread* bekerja.




Gambar 2.13. Proses multi *Thread*

Sumber : Ady Wicaksono. *Pemrograman Internet dan XML pada Ponsel dengan MIDlet Java,* (Jakarta: Elex Media Komputindo, 2002), p.83

Sebelum membuat suatu *Thread*, kita harus mengerti daur hidup suatu *Thread* seperti yang ditampilkan Gambar 2.14 berikut :




Gambar 2.14. Daur Hidup *Thread*

Sumber : Ady Wicaksono. *Pemrograman Internet dan XML pada Ponsel dengan MIDlet Java,* (Jakarta: Elex Media Komputindo, 2002), p.87

Cara membuat *Thread* di Java mengacu pada fungsi `run()`. Pada fungsi ini kita busa memasukkan statement Java yang menentukan bagaimana *thread* nanti berjalan. Dua cara yang digunakan membuat *thread*, yaitu :

1. Membuat kelas turunan dari *Thread* dan mengoverride fungsi `run()`.
2. Mengimplementasikan *interface* Runnable, dimana fungsi `run()` harus diimplementasikan dan harus dibuatkan objek *Thread* yang mengacu ke kelas yang dimaksud.

Berikut contoh implementasi *Thread* menggunakan metode Runnable :

```
class exThread implements Runnable {
      Thread t;
```

```
public exThread (String str) {
      t = new Thread(this, str);
      t.start(); }

// konstruktor
public void run () {
      for ( int i=0; i<10 ; i++ )
            { System.out.println(i + " " + t.getName());
            try {
                  t.sleep((long) (Math.random() * 1000)); }
            catch ( InterruptedException e ) {} }
            System.out.println("DONE" +t.getName());
            } }

public class TwoThread {
      public static void main (String[] args) {


            new exThread("Haloo");
            new exThread("Apa kabar");
            } }
      }
```

Pada program di atas, dibuat dua *Thread* yang masing-masing melakukan penulisan kata 'Haloo' dan 'Apa kabar' sebanyak 10 kali tiap waktu random. Proses penulisan tidak menunggu proses yang satunya selesai namun bersama-sama dikerjakan.

1. Ketika fungsi `start()` diinisialisasi, secara otomatis sistem akan menjadwalkan dan menjalankan fungsi `run()` pada *thread*. Dari keadaan running tersebut, *thread* dapat mengalami dua hal yaitu stop atau blocked. Stop terjadi karena terjadi terminasi pada fungsi `run()`. Sedangkan blocked bisa terjadi karena hal-hal berikut :Dilakukan pemanggilan terhadap fungsi `sleep()` dari objek *Thread*.
2. Dilakukan pemanggilan terhadap fungsi `yield()` dimana *Thread* akan berhenti sementara waku.
3. Ada statement untuk operasi I/O, seperti pembacaan data dari keyboard, pengambilan data dari network, dan sebagainya.

Berikut fungsi-fungsi kelas *Thread* pada lingkup J2ME :

- `public static void start ()`

  Fungsi ini mengaktifkan *Thread*, sekaligus menjalankan fungsi `run()`.
- `public static Thread currentThread()`

  Fungsi ini menghasilkan reference ke objek *Thread* yang sedang aktif.

- `public static void sleep (int milisec)`
  Fungsi ini membuat *Thread* sementara menjadi tidak tidak aktif selama ukuran variable milisec dalam seperseribu detik.
- `public void run()`
  Fungsi ini menjalankan *Thread* dan dipanggil otomatis oleh JVM setelah pemanggilan fungsu `start()`.

Pada paket `java.util` tersedia kelas `Timer` dan `TimerTask` yang berguna untuk mengeksekusi suatu tugas dengan background *Thread*. Kerjanya mirip dengan *Thread* karena sama-sama mengimplementasikan *interface* Runnable yang mana bila aktif akan menjalankan fungsi public void `run()`. Kelas `Timer` dan `TimerTask` berguna untuk mengeksekusi suatu tugas dengan background *Thread*. Perhatikan kode berikut :

```
...

Timer timer = new Timer();
TimerTask task = new ExampleTask();
TimerTask task2 = new ExampleTask2();

...

timer.schedule(task,10000);
timer.schedule(task2,500,500);

...

class ExampleTask extends TimerTask {
      public void run()
            {System.out.println(“1”); }

class ExampleTask2 extends TimerTask {
      public void run()
            {System.out.println(“2”);
            ...   }
```

Ketika `Timer` digunakan, maka kelas `TimerTask` juga harus diinisialisasi, karena saat penggunaan `Timer`, maka bila waktu yang ditentukan telah terjadi, maka secara otomatis akan menjalankan suatu task pada kelas turunan `TimerTask` fungsi `run()`. Pada program di atas, `Timer` menjalankan dua interval waktu yaitu setiap 500 milidetik menjalankan ExampleTask dan menjalankan `ExampleTask2` setelah 10000 milidetik timer dijalankan.

Fungsi-fungsi `Timer` yang bisa digunakan untuk menjalankan kelas TimerTask sesuai jadwal yang dibuat, antara lain :

- `public void schedule (TimerTask t, long delay)`

  Kelas `TimerTask t` akan dieksekusi terhitung setelah jeda waktu delay pada parameter dihitung dalam satuan milidetik.
- `public void schedule (TimerTask t, long delay, long period)`

  Kelas `TimerTask t` akan dieksekusi terhitung setelah jeda waktu delay pada parameter dihitung dalam satuan milidetik dan akan diulangi lagi pada jeda waktu period (milidetik) terhitung setelah waktu terakhir `TimerTask t` dijalankan.
- `public void schedule (TimerTask t, Date fisttime, long period)`

  Kelas `TimerTask t` akan dieksekusi pada waktu yang sesuai dengan parameter Date firsttime yang ada dan akan diulangi lagi pada jeda waktu `period (milidetik)` terhitung setelah waktu terakhir `TimerTask t` dijalankan
- `public void cancel ()`

  Fungsi ini digunakan untuk terminasi atau menghentikan timer, sehingga semua task pada TimerTask tidak akan dieksekusi lagi.

Keseluruhan dasar teori dan penggunaan kode pemrograman menggunakan bahasa Java yang telah dipaparkan pada bab-bab sebelumnya, pemaparannya tidak selengkap-lengkapnya. Untuk membatasi pembahasan, teori yang dituliskan hanya yang digunakan dalam pembuatan program game *multiplayer* ini.

## 2.5. *Networking* MIDlet menggunakan HTTP *Connection*

Pada aplikasi yang akan dibuat, protokol yang digunakan untuk komunikasi antara *client* ke *server* menggunakan HTTP. *Server* yang menjalankan HTTP biasa disebut sebagai *web server*. Protokol HTTP merupakan protokol yang dikembangkan untuk transfer dokumen dengan format HTML. Namun demikian pada perkembangannya, HTTP juga sering dipakai untuk transfer berbagai format data baik data *audio* sampai dengan data *visual*. Protokol HTTP bekerja di atas protokol TCP/IP dengan nomor *port* 80.

Penggunaan koneksi HTTP untuk komunikasi dengan jaringan (*internet*) menawarkan beberapa keuntungan utama:

- Tidak semua perangkat handphone yang menggunakan MIDP, mendukung fasilitas *socket* dan *datagram* untuk komunikasi, sedangkan semua MIDP pasti mendukung komunikasi menggunakan HTTP.
- Koneksi *socket* dan *datagram* bergantung sekali dengan jaringan operator yang digunakan. Tidak jarang, operator hanya menerapkan satu jenis komunikasi saja, sehingga dengan pembatasan ini, aplikasi *wireless* yang dibuat tidak leluasa penggunaannya.
- Akibat dukungan wajib protokol HTTP pada MIDP, menjadikan aplikasi *wireless* yang dibuat menggunakan koneksi HTTP sangat fleksibel dengan perubahan jaringan *wireless* lain.

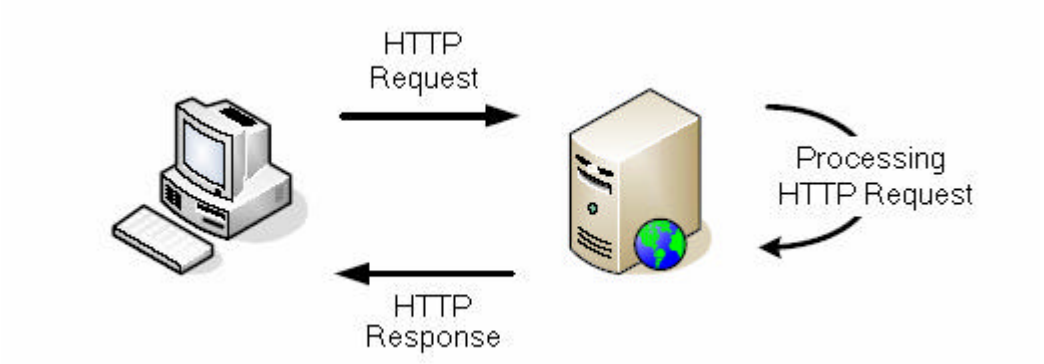
Implementasi dari protokol HTTP atau sebuah aplikasi yang mendefinisikan kerja sebuah protokol HTTP terbagi menjadi 2 bagian yaitu :

1. HTTP *Client*

   Terdiri atas aplikasi-aplikasi yang mengimplementasikan protokol HTTP sebagai *client*, misalnya Internet Explorer, Nestcape dan sebagainya..

2. HTTP *Server*

   Terdiri atas atas aplikasi yang mengimplementasikan protokol HTTP sebagai *server* misalnya Apache *Web Server*, Microsoft Internet Information *Server* (IIS).




Gambar 2.15. Cara Kerja Protokol HTTP

Penjelasan dari gambar di atas adalah sebagai berikut :

1. HTTP *client* akan melakukan koneksi ke HTTP *server* dan jika koneksi sudah berhasil dilakukan, maka HTTP *client* akan mengirimkan *request*.

2. HTTP *server* (yang menerima *request* yang diminta oleh HTTP *Client*) akan memprosesnya dan mengirimkan *response* ke HTTP *client*.
3. Saat *response* telah selesai dikirimkan, maka koneksi akan diputus oleh HTTP *server*.

Melihat gambaran di atas, protokol HTTP merupakan protokol yang *stateless*, maksudnya setelah *web server* mengirimkan respon, maka koneksi akan diputuskan.

MIDP menyediakan fasilitas khusus untuk koneksi dengan eksternal resource melalui fungsionalitas khusus *GenericConnection Frameworks* (GCF). GCF dari CLCD digunakan untuk mendukung *client-server*, dimana implementasi MIDP harus menyediakan dukungan untuk mengakses *server* dan layanan HTTP.

CGF dari CLDC menyediakan base stream dan content *interface*. *Interface* HttpConnection menyediakan fungsionalitas tambahan yang diperlukan unutk menetapkan header, *parse response headers* dan berbagai fungsi spesifik HTTP. GCF mendeskripsikan sebuah kelas dasar yaitu *Connector* dan beberapa *interface* yang dipakai untuk menyelenggarakan semua koneksi ke jaringan, berada pada package javax.microedition.io.

*Interface* yang biasa digunakan untuk koneksi ke *server* HTTP adalah menggunakan `HttpConnection` dengan membuka dan menutu koneksi dasar *interface* `Connection`. Contoh koneksi dengan menggunakan *interface* HttpConnection, yaitu:

```
HttpConnection conn = null;
conn = (HttpConnection)
      Connector.open("http://202.43.253/halo.php);
```

Setelah koneksi terselenggara, harus digunakan kelas I/O untuk mulai membaca atau menulis ke dan dari koneksi. Kelas I/O yang biasa digunakan antara lain:

- `InputStream dan OutputStream`

  Kelas dasar untuk semua aliran (stream) data yang masuk atau keluar
- `InputStreamReader`

  Stream tempat data dapat dibaca sebagai karakter teks.
- `Reader`

  Kelas abstrak untuk membaca stream karakter.

- `Writer`

  Kelas abstrak untuk menulis stream karakter.

Berikut contoh program koneksi ke jaringan menggunakan HttpConnection :

```
HttpConnection c = null;
InputStream is = null;
try {
      c = (HttpConnection)Connector.open
            ("http://202.43.253.52/teks.html);
      is = c.openInputStream();
      if (is!= null) { is.close(); }
      if (c!= null) { c.close(); }
} catch (IOException e) { }
```

Setelah `Connector` dibuka, koneksi dan request dilakukan ke alamat yang dimaksud. Isi dari page HTML tersebut, nantinya dibaca sebagai karakter teks yang diterima oleh variable `is` sebagai `InputStream`-nya. Bila alamat yang diminta tidak ada, maka koneksi akan diputus dengan memanggil fungsi `close()`, demikian pula bila isi page tidak ada, maka `InputStream` akan ditutup pula.

### 2.6. PHP dan *Structured Query Language* (SQL)

PHP atau resminya bernama PHP: *Hypertext PreProcessor* merupakan suatu bahasa pemrograman berbasis *web* yang bersifat dinamis. *Script* PHP bersifat *server side* dan ditambahkan ke dalam *file* HTML. Sifat *server side* berarti pengerjaan *script* akan dilakukan di *server*, baru kemudian hasilnya dikirim kepada *browser*. Keunggulan dari sifat *server side* ini antara lain :

- Tidak memerlukan kompatibilitas browser atau tidak harus menggunakan *browser* tertentu. Hal ini disebabkan karena *server* yang akan mengerjakan *script* PHP. Hasil yang dikirimkan pada *browser* umumnya bersifat gambar atau teks sehingga pasti dikenali oleh *browser* apapun.
- Dapat memanfaatkan sumber-sumber aplikasi yang dimiliki oleh *server* misalnya koneksi ke *database*.
- Skrip tidak dapat dilihat menggunakan fasilitas view HTML source karena yang dikirimkan ke *user* adalah hasil pemrosesan di *server*.

Selain keunggulan dari *server* side, PHP juga dapat melakukan semua aplikasi program CGI seperti mengambil nilai *form*, menghasilkan halaman *web* yang dinamis serta mengirim dan menerima *cookie*.Selain itu, salah satu faktor utama yang membuat PHP menjadi sangat populer dalam pembuatan aplikasi berbasis

*web* dan situs *web* dinamis adalah karena bahasa ini mendukung demikian banyak *database.*

Database yang didukung oleh PHP salah satunya adalah MySQL, yang digunakan pada pembuatan game ini. MySQL sendiri merupakan perangkat lunak yang open source. Bahasa standar yang digunakan oleh MySQL adalah SQL (*Stuctured Query Language)* yang mana merupakan bahasa standar pengolahan database, dimana PHP juga menggunakannya. Beberapa perintah dasar SQL pada PHP antara lain :

- *Insert*

  Digunakan untuk memasukkan suatu data ke dalam tabel

  Sintaks : `INSERT INTO namatabel VALUES data;`
- *Select*

  Digunakan untuk mengambil data dari suatu tabel

  Sintaks : `SELECT * FROM namatabel WHERE kondisi;`

  `SELECT namafield FROM namatabel WHERE kondisi;`

  Tanda * menunjukkan bahwa nilai semua *field* yang ada akan dipilih. Sedangkan jika menggunakan nama *field*, maka hanya nama field yang didefinisikan yang diambil nilainya.
- *Update*

  Digunakan untuk mengedit atau memperbaharui nilai suatu data

  Sintaks : `UPDATE namatabel SET kriteria WHERE kondisi;`
- *Delete*

  Digunakan untuk menghapus suatu *record* dari tabel

  Sintaks : `DELETE FROM namatabel WHERE kondisi;`

## 2.7. J2ME untuk *Wireless Games*

Menyimak perkembangan game, pada awal mulanya hanya digunakan sebagai pelengkap fitur tambahan dari handphone. Karakteristik game-game yang dibuat menggunakan masih sederhana. Karena perkembangan dari Java dimana J2ME diterbitkan untuk kepentingan membuatan aplikasi di handphone, ikut mendorong perkembangan game berbasis Java.

Suatu *survey* tentang penjualan *video game* di dunia, dimana menghasilkan 28 miliar Dollar Amerika pada tahun 2001, dengan penjualan *game wireless* mencapai 700 juta Dollar Amerika, diperkirakan meningkat menjadi 30 miliar Dollar Amerika pada tahun 2006, dengan penjualan game *wireless* mencapai 3,6 milyar . Peningkatan juga terjadi pada jumlah *user* atau pemain dimana pada tahun 2002 diperkirakan terdapat 7 juta pemain *game wireless* yang akan naik menjadi 71,2 juta *user* tahun 2007.

Melihat hasil *survey* tersebut, bisa diambil pandangan bahwa aplikasi *game wireless* masih sangat potensial untuk dikembangkan, salah satunya dengan penggunaan J2ME sebagai *platform game.*

Game *wireless* berbasis Java 2 ME memiliki tantangan dan karakteristik yang unik karena banyak dipengaruhi oleh latency pada jaringan dan kekuatan perangkat itu sendiri. Pembuatannya diusahakan sesederhana mungkin, user-friendly, dan mudah untuk dikembangkan ke depan. Adapun hal-hal yang menjadi batasan pengembangan game *wireless* berbasis Java yang utama, antara lain:

- Ruang penyimpanan (*memory*) baik yang statik maupun dinamik
- Besar layar tampilan (*screen*)
- Kemampuan unit pemrosesan (CPU) pada handphone
- *Bandwidth* atau kecepatan aliran data
- *Latency* atau waktu bolak-balik yang dibutuhkan suatu data untuk sampai ke tujuan dan kembali lagi ke asal.