BAB 3

CLASS GameCanvas

1.1 Tujuan:

Setelah mempelajari modul ini peserta diharapkan dapat:

- Mengenal Kelas *GameCanvas*
- Implementasi Sederhana dari Kelas *GameCanvas*

1.2 Mengenal Kelas *GameCanvas*

Kelas *GameCanvas* menyediakan fungsi-fungsi penting dalam menangani antar muka game. Merupakan turunan dari kelas *Canvas (commands, input events, etc.)*, dengan penambahan fitur antara lain kemampuan game yang spesifik seperti: *off-screen graphic buffer* dan kemampuan untuk mengetahui status *key-key input* dari key-key pada HP.

Kelas *GameCanvas* pada MIDP 2.0 telah mengusung fitur-fitur penting yang mengatasi masalah yang ada pada MIDP 1.0. Yaitu sekarang kita cukup menggunakan sebuah (*single) buffer* untuk setiap obyek *GameCanvas* yang dibuat. Ini sangat penting karena tidak hanya meminimalkan penggunaan *heap memory* yang digunakan oleh game. Tetapi juga agar game bisa dikontrol dengan hanya menggunakan sebuah *(single) loop*.

Untuk sekedar diketahui, berikut cara lama *looping* sebuah game (MIDP 1.0):

```
public void theGameCanvas extends Canvas implements Runnable {
 public void run() {
  while (true) {
  repaint();  // update tampilan game pada display
  }
}
public void paint(Graphics g) {
  // painting // redraw occurs here
 }
protected void keyPressed(int keyCode) {
  // obtain user inputs
 }
}
```

Seperti kita lihat pada kode di atas ada tiga bagian fungsi berbeda yaitu, bagian penggambar (*painting*) pada *screen*, bagian *run()*, dan pembaca *input key*. Ketiganya berjalan pada masing-masing *thread* yang berbeda. Karena *thread* yang berbeda itulah terkadang ada jeda dan kelambatan respon terutama terlihat pada jenis game *arcade/action* yang membutuhkan banyak permainan *graphic* dan interaktifitas tinggi.
Namun sekarang, pada MIDP 2.0 dan implementasi *GameCanvas,* semua kode tampak bersih, lebih mudah digunakan dan efisien. Yaitu karena menggunakan *single thread*, *GameCanvas* tidak lagi menunggu sebuah *event keyPressed* tetapi sudah diganti dengan teknik *polling*, artinya kita bisa membaca *key-key* mana yang ditekan pada suatu saat dengan menggunakan method *getKeyState()* yang sudah disediakan oleh *GameCanvas*. Karena menggunakan teknik *buffering* pada *GameCanvas*, yang disebut *double buffering*, penggambaran pada *screen/display* dilakukan otomatis (tanpa mendefinisikan sebuah *thread* tersendiri). Kita hanya perlu memanggil *method flushGraphics()* untuk menampilkan *graphics* ke *display*. Teknik *Double Buffering* tersebut dimaksudkan untuk menghindari *flickering* (efek berkedip) pada *display* dengan cara menggambar *graphics* ke *file image* sementara (*temporary*) secara *off set* (di *background*), dan akan ditampilkan ke *display* begitu *file image* sudah terisi *graphics* lengkap.

1.3 Implementasi Sederhana dari Kelas *GameCanvas*

```
public class ExampleGameCanvas extends GameCanvas implements Runnable {
      // Constructor and initialization
       public ExampleGameCanvas() {
        super(true);
        ...
      }

      // Automatically start thread for game loop
      public void start() {
            isPlay = true;
            Thread t = new Thread(this);
            t.start();
      }

      public void stop() { isPlay = false; }
      // Main Game Loop
      public void run() {
            Graphics g = getGraphics();
            while (isPlay == true) {
```

```
            input();
            drawScreen(g);
            try { Thread.sleep(delay); }
            catch (InterruptedException ie) {}
        }
      }

      // Method to Handle User Inputs
       private void input() {
        int keyStates = getKeyStates();

        // Left
        if ((keyStates & LEFT_PRESSED) != 0)
        currentX = Math.max(0, currentX - 1);
        ...
      }

      // Method to Display Graphics
      private void drawScreen(Graphics g) {
        ...
        flushGraphics();
      }
}
```