PEMOGRAMAN APLIKASI BERGERAK (Java ME)

Yoannita

COMMAND

Event Handling with Commands

Overview

- Displayable, the parent of all screen displays, supports a very flexible user interface concept, the command.
- A *command* is something the user can invoke—you can think of it as a button.
- Like a button, it has a title, like "OK" or "Cancel," and your application can respond appropriately when the user invokes the command.
- The premise is that you want a command to be available to the user, but you don't really care how it is shown on the screen or exactly how the user invokes it—keypad button, soft button, touch screen, whatever.

Overview

- Semua kelas Displayable menyimpan command-command yang digunakannya dalam suatu daftar/list
- Anda dapat menambah atau membuang command menggunakan method berikut ini :

  public void addCommand(Command cmd)

  public void removeCommand(Command cmd)

Creating Commands

- Pada MIDP, semua command direpresentasikan sebagai instance dari kelas Command.
- Untuk menciptakan suatu command, anda perlu mengisi nama, tipe dan priority dari command tersebut.
- Nama command biasanya tampil pada layar. Tipe command dapat digunakan untuk mendeskripsikan kegunaan dari command tersebut.

Command Types

- Implementasi posisi Command (*softkey* yang merepresentasikan) ini tergantung pada vendor perangkatnya. Berikut tipe-tipe dari command :

NAME	MEANING
OK	Confirms a selection.
CANCEL	Cancels pending changes.
BACK	Moves the user back to a previous screen.
STOP	Stops a running operation.
HELP	Shows application instructions.
SCREEN	Generic type for specific application commands.

Ringkasan Constructor

1. Command (String label, int commandType, int prioritas)

   → Menciptakan command baru objek dengan label yang diberikan pendek, jenis, dan prioritas.

2. Command (String shortLabel, String longLabel, int commandType, int prioritas)

   → Menciptakan command baru objek dengan label yang diberikan, jenis, dan prioritas.

Membuat sebuah Command

- Command(String label, int commandType, int priority)

- Contoh pendeklarasian :

```
Command exitCommand = new
  Command("Exit", Command.EXIT, 1);
```

Command Parameter

- Contoh 2:
- Command infoCommand = new Command("Info", Command.SCREEN, 2);

- **Label** : "Info" ← tulisan yang akan muncul ke user
- **Type** : "Command.SCREEN"

  Ada beberapa tipe command : BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, and STOP.
- **Priority** : "2" →

Priority

- Priority menentukan urutan prioritas penampilan suatu Command pada layar. Semakin kecil priority maka Command itu akan lebih mudah diakses dibandingkan dengan Command yang nilai priority -nya lebih besar.

Priority

- Contoh :
  - readCommand = new Command("Read", Command.ITEM, 1);
    replyCommand = new Command("Reply", Command.ITEM, 2);
    delCommand = new Command(“Delete", Command.ITEM, 3);

  - list.addCommand(readCommand);
  - list.addCommand(replyCommand);
  - list.addCommand(delCommand);

Urutan yang akan ditampilkan ke user :

Priority

- Contoh :
- readCommand = new Command("Read", Command.ITEM, 1);
  replyCommand = new Command("Reply", Command.ITEM, 2);
  delCommand = new Command(“Delete", Command.ITEM, 3);

- list.addCommand(delCommand);
- list.addCommand(replyCommand);
- list.addCommand(readCommand);

- Tampilan:

Priority

- Contoh:
- readCommand = new Command("Read", Command.ITEM, 1);
- delCommand = new Command("Delete", Command.ITEM, 1);
- replyCommand = new Command("Reply", Command.ITEM, 1);

- list.addCommand(delCommand);
- list.addCommand(replyCommand);
- list.addCommand(readCommand);

- Tampilan:

Command

- Command dapat dimasukkan ke dalam obyek Displayable mengunakan *method* sebagai berikut :
  - public void addCommand(Command cmd)
- Sedangkan penghapusan dengan menggunakan *method* sebagai berikut :
  - public void removeCommand(Command cmd)
- Jangan lupa untuk menambahkan setCommandListener() pada Form/Canvas/List yang menggunakan CommandAction

CommandListener

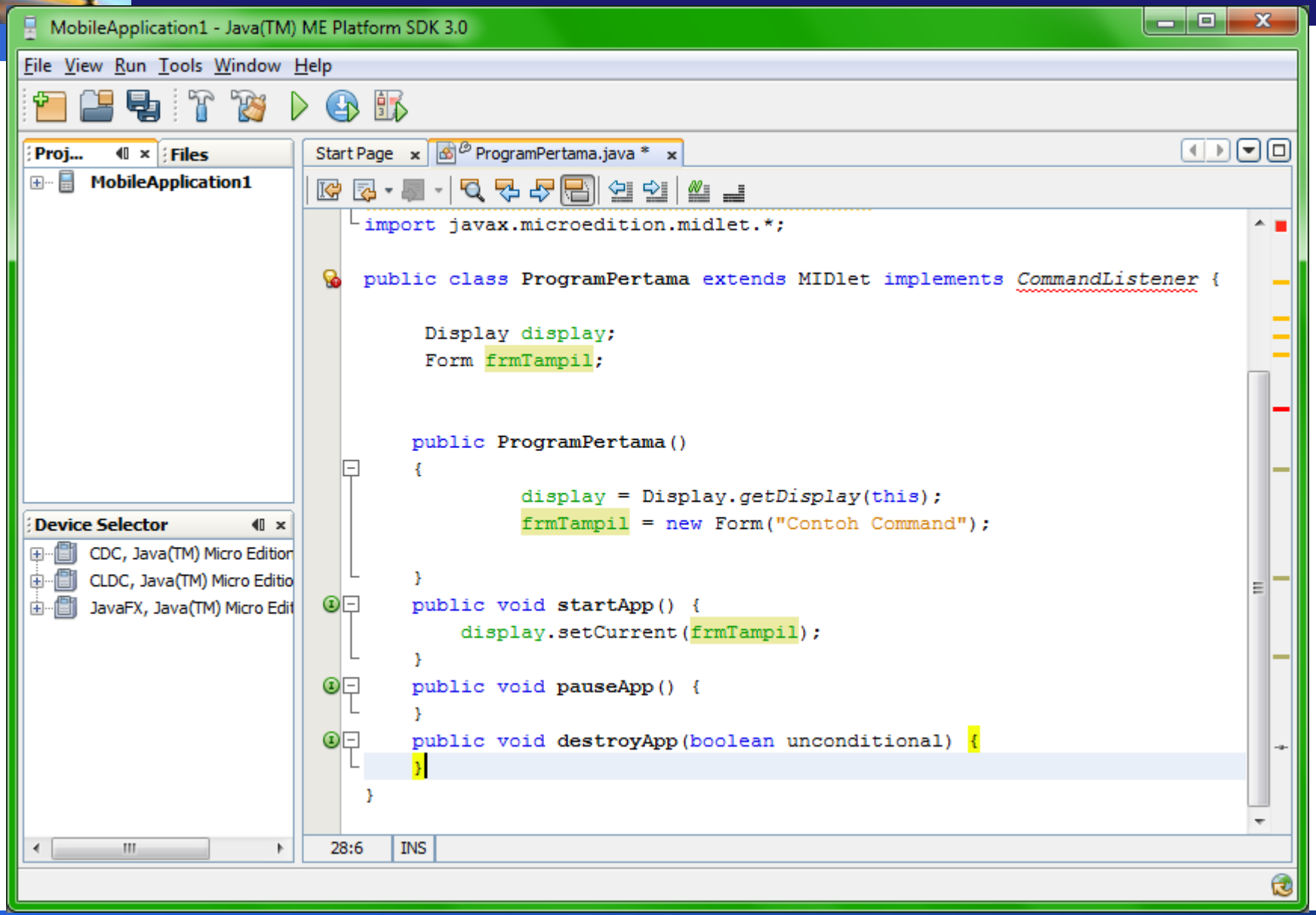
- CommandListener merupakan interface dengan single method:

```
void commandAction(Command command, Displayable displayable)
```

- Method commandAction() akan dipanggil jika Command dipilih. Variabel command merupakan referensi Command yang telah dipilih.
- Oleh karena itu, sintaks setCommandListener perlu diberikan kepada objek yang menggunakan/menambahkan command (addCommand).
- Contoh :

```
frmSMS.addCommand(cmdExit);
frmSMS.setCommandListener(this);
```

Langkah-langkah



Langkah-langkah

Tambahkan tulisan **implements CommandListener** setelah extends MIDlet

```
public class ProgramPertama extends MIDlet implements CommandListener {

    Display display;
    Form frmTampil;

   public ProgramPertama()
   {
          display = Display.getDisplay(this);
          frmTampil = new Form("Contoh Command");

   }
```

Langkah-langkah

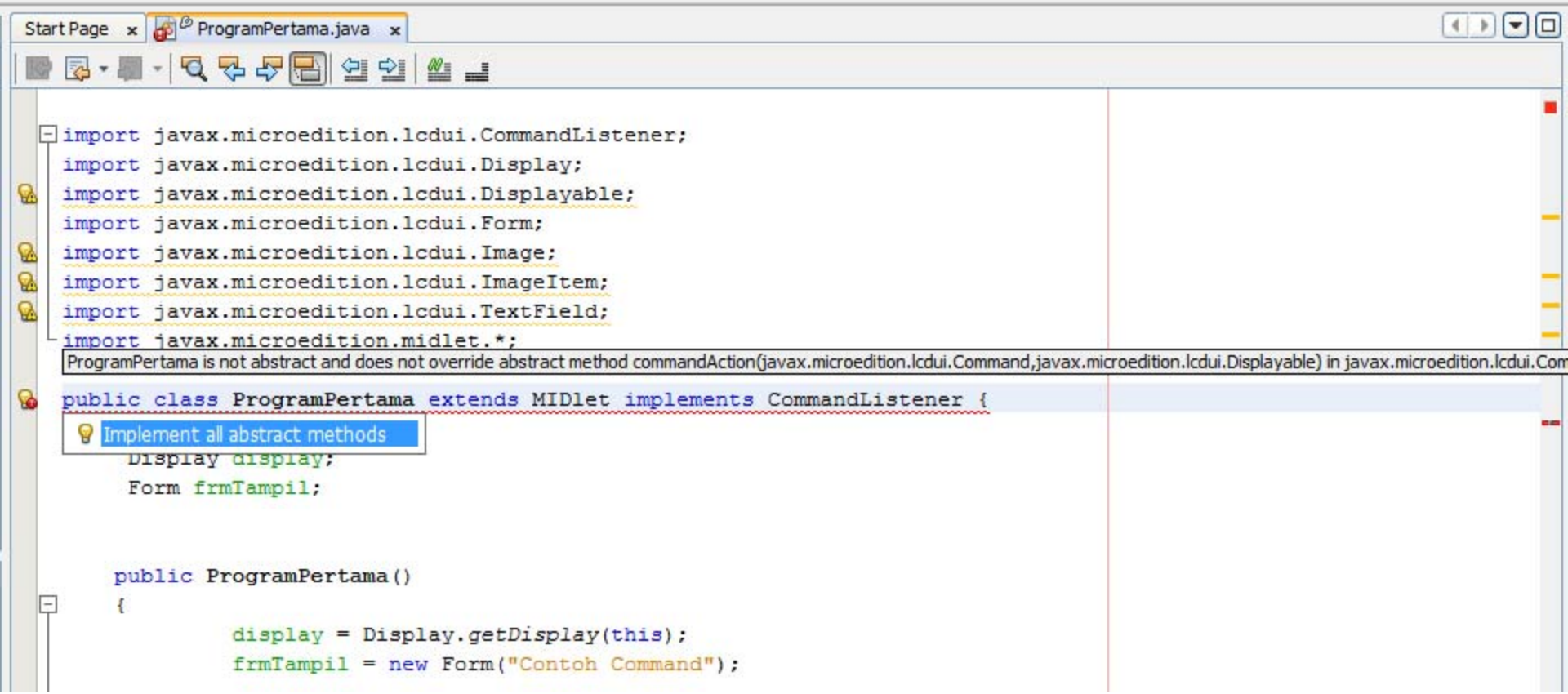
- Akan muncul warning/error,
- arahkan kursor ke tanda lampu, lalu **pilih import**,

```
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.TextField;
cannot find symbol                    ition.midlet.*;
symbol: class CommandListener

public class ProgramPertama extends MIDlet implements CommandListener {
    Add import for javax.microedition.lcdui.CommandListener
    Create interface "CommandListener" in package
    Form frmTampil;
```

Langkah-langkah

- setelah itu **arahkan lagi** kursor ke tanda lampu**, pilih implements all abstract methods**



❖Tambahkan sintaks program berikut ini :

```
public class ProgramPertama extends MIDlet implements CommandListener {

    Display display;
    Form frmTampil;
    Command cmdKeluar;

    public ProgramPertama()
    {
            display = Display.getDisplay(this);
            frmTampil = new Form("Contoh Command");
            cmdKeluar = new Command("Keluar", Command.EXIT, 1);
            frmTampil.addCommand(cmdKeluar);
            frmTampil.setCommandListener(this);
    }
```

- Tambahkan baris perintah berikut ini ke dalam method commandAction() :

```
if (c == cmdKeluar)
        {
           destroyApp(false);
          this.notifyDestroyed();
        }
```

```
    public void startApp() {
        display.setCurrent(frmTampil);
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable d) {
        if (c == cmdKeluar)
        {
            this.notifyDestroyed();
        }
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

32:46 INS

Keterangan

- Pada method destroyApp( ), MIDlet harus melepaskan semua resource yang dialokasikan kepadanya, memutuskan semua threads yang berjalan, dan menghentikan semua timer yang aktif.
- Ketika MIDlet diakhiri seperti ini, isi argumen dari destroyApp() bernilai true, untuk mengindikasikan bahwa MIDlet tidak dapat mencegah proses untuk terus berlanjut.
- Namun, dalam beberapa keadaan tertentu, adalah berguna untuk memberikan pilihan pada MIDlet untuk tidak mengakhiri proses, misalkan jika kemungkinan ada data yang dibutuhkan untuk disimpan, dalam hal ini method destroyApp() dapat dipanggil dengan argumen false, yang menunjukkan bahwa MIDlet dapat dilanjutkan dengan melempar kesalahan MIDletStateChangeException

Keterangan

- Kode program berikut ini mengilustrasikan bagaimana teknik yang baik untuk mengimplementasikan kondisi shutdown pada suatu MIDlet :

```
try {
    // Call destroyApp to release resources
    destroyApp(false);

    // Arrange for the MIDlet to be destroyed
    notifyDestroyed(  );

} catch (MIDletStateChangeException ex) {
    // MIDlet does not want to close
}
```

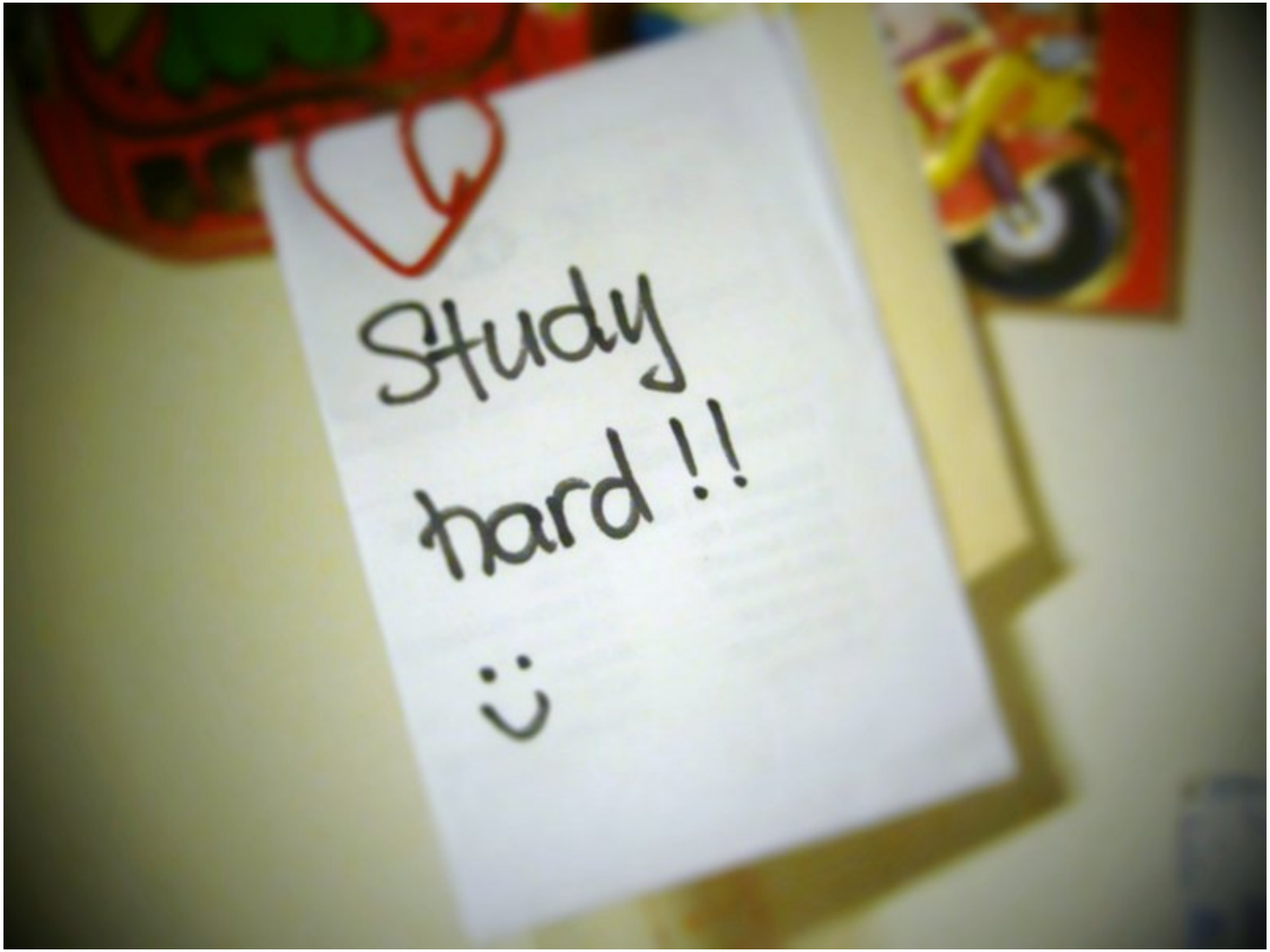
onjava.com

Keterangan

- This code might be used to respond to an Exit button in the MIDlet's user interface.
- It begins by directly invoking the MIDlet's own destroyApp( ) method so that resources are released.
- If the MIDlet is not in an appropriate state to terminate, anddestroyApp( ) is called with argument false, the MIDlet should throw a MIDletStateChangeException.
- The calling code should catch this exception and do nothing, as shown here.
- On the other hand, if the MIDlet is prepared to be terminate, it should complete the destroyApp( ) method normally, in which case the calling code uses the MIDlet notifyDestroyed( )method to tell the MIDP platform that the MIDlet wants to be terminated.

Keterangan

- When the MIDlet is being destroyed by the platform, most likely because the user has requested it, the MIDlet'sdestroyApp( ) method is called with the argument true, and the MIDlet is destroyed when this method completes. It is not necessary in this case for the MIDlet to invoke its notifyDestroyed( ) method.
- When the MIDlet itself wants to terminate, typically because it has no more useful work to do or the user has pressed an Exit button, it can do so by invoking its notifyDestroyed( ) method, which tells the platform that it should be destroyed.
- In this case, the platform does not call the MIDlet's destroyApp() method; it assumes that the MIDlet is already prepared to be terminated. Most MIDlets invoke their own destroyApp( ) method to perform the usual tidy up


Study
hard !!


Thanks!