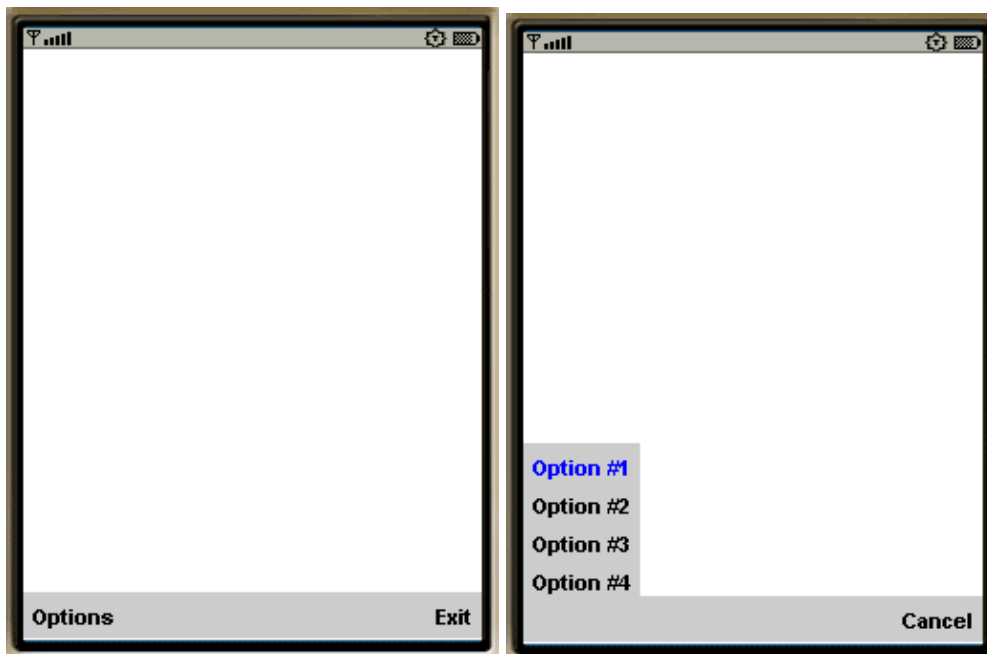


Contents

- [1 Overview](#)
- [2 Architecture](#)
- [3 Drawing the menu](#)
- [4 Option selecting logic](#)
- [5 The source code](#)
 - ◆ [5.1 Midlet.java](#)
 - ◆ [5.2 GUI.java](#)
 - ◆ [5.3 Menu.java](#)

Overview

This example will show you how to create an advanced menu in mobile Java applications using the low-level graphical API:



The solution presented here consists of three Java classes: the Midlet, the GUI and the Menu. The Menu is drawn at the bottom of mobile screen, and in the inactive state mobile user can see only the menu bar with default menu items: "Options" and "Exit". When "Options" item is selected (by pressing the left soft-key on the phone) the menu goes into the active state and draws all menu options on the screen.

The code presented here is quite short and might be customized in many different ways. However, it is far, far away from the optimized one, and its purpose was only to show the principles of drawing highly customized menu by using the low-level graphical API.

Architecture

The MIDlet class simply initializes the GUI and shows it to the mobile user.

The GUI class keeps inside a Menu object, and changes the state of Menu according to the user's inputs.

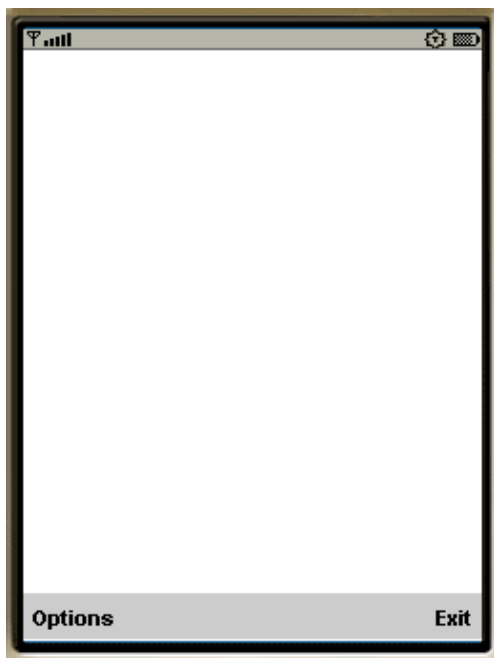
The Menu class after all simply draws itself on a canvas according to its current state.

Drawing the menu

The main thing about drawing the Menu is just to remember that it should be (re)drawn according to its current state, and state changes are made by the user through the "UP", "DOWN", "SELECT" ("LEFT" and "RIGHT") buttons.

This simple menu can be only in two states: active and inactive.

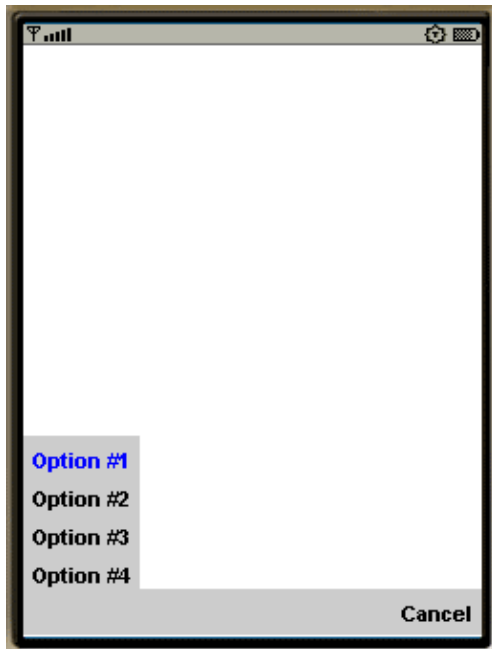
In the inactive state the menu bar is displayed only:



Here you can see two major options: the "Options" for activating the menu; and "Exit" for exiting the MIDlet.

The inactive menu is drawn by the method drawInactiveMenu(), which you can find from the Menu.java

The active state means drawing menu options on the top of menu bar, and also changing the menu bar's texts:

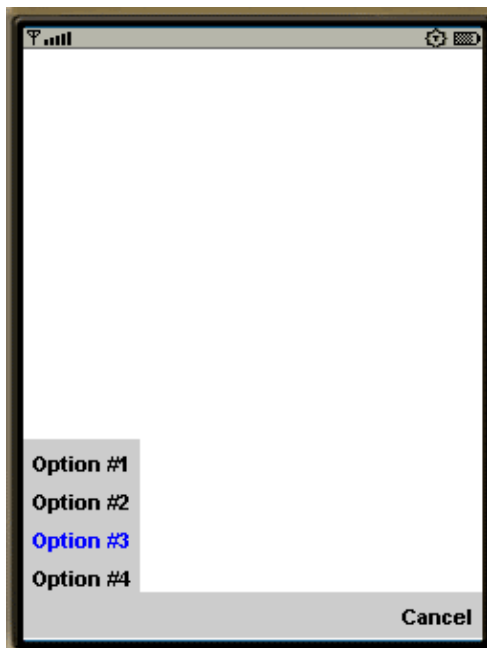


"Option #1" will be always selected by default. Note, that instead of "Exit" option here we draw "Cancel" option. This is the usual way of how the menu logic is implemented. This, however, could be customized in many ways.

The active menu is drawn by the method `drawActiveMenu()`, which you can find from the `Menu.java`

Option selecting logic

Now you can press "UP" and "DOWN" navigations buttons in order to make your choice:

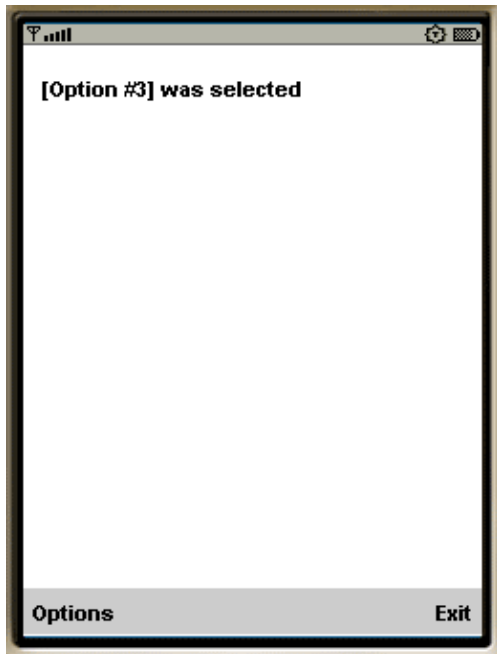


How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

The GUI class traces user inputs through the `keyPressed()` method, and changes the state of Menu. The logic hidden in the `keyPressed()` method could be placed inside the Menu class, but to make code more easier to understand, the Menu class keeps only the graphical methods.

The option selecting graphics could be implemented in many different ways, but the simplest one is either underlining option's text or changing its color.

When option is chosen, press "Select" or "FIRE" button and the GUI class will notify Menu about the user's choice:



After pressing the "Select" button the menu state will be changed to inactive, and the `drawInactiveMenu()` method will be called.

If user will press the "Cancel", then no option will be selected and inactive menu will be drawn once again.

The source code

Please note, that the code presented here is far, far, far away from the optimized one. The reason for implementing everything in separated methods was to give the developer a possibility to easily change the code and see immediately how changes are affecting on GUI and its look-and-feel.

Also the option selecting logic should be stored inside the Menu class, but for making code more simpler, it remains inside the GUI class.

I hope now you have the basic idea of how the advanced menu could be implemented in the J2ME applications using the low-level graphical API.

Midlet.java

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

public class Midlet extends MIDlet {

    private Display display;
    private GUI gui;

    public void startApp() {

        display = Display.getDisplay(this);

        if (display == null) {

            destroyApp(false);

        } // end if display is not allocated

        gui = new GUI(this);

        display.setCurrent(gui);

        gui.start();

    } // end startApp

    public void pauseApp() {
    } // end pauseApp

    public void destroyApp(boolean unconditional) {

        display.setCurrent(null);

        notifyDestroyed();

    } // end destroyApp

} // end Midlet.java
```

GUI.java

```
import java.util.*;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

import javax.microedition.media.*;
import javax.microedition.media.control.*;

import java.io.*;
import javax.microedition.io.*;
```

How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

```
import javax.microedition.io.file.*;

class GUI extends GameCanvas {

    private Midlet midlet;
    private Display display;
    private Graphics g;
    private Font font;

    private int width = 0;
    private int height = 0;

    private Menu menu;

    private String leftOption;
    private String rightOption;
    private String[] menuOptions;

    private int currentlySelectedIndex = 0;

    private boolean menuIsActive = false;

    /**
     * Creates a new instance of GUI.
     */
    public GUI(Midlet midlet) {

        super(false);

        this.midlet = midlet;

        font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_SMALL);

        setFullScreenMode(true);

        width = getWidth();
        height = getHeight();

        g = getGraphics();

        leftOption = "Options"; // will be displayed only when menu is not active
        rightOption = "Exit"; // will be displayed only when menu is not active
        menuOptions = new String[]{ "Option #1", "Option #2", "Option #3", "Option #4"};

        menu = new Menu(leftOption, rightOption, menuOptions);
    } // end constructor

    public void start() {

        clearScreen();

        menu.drawInactiveMenu(this, g);
    } // end start

    // softkey codes may vary from phone to phone
    // -6 and -7 values are OK on Nokia phones
}
```

How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

```
private int LEFT_SOFTKEY_CODE = -6; // check it for your phone model
private int RIGHT_SOFTKEY_CODE = -7; // check it for your phone model

protected void keyPressed(int keyCode) {

    // work with menu according to its current state

    if (menuIsActive) { // draw active menu

        if (keyCode == RIGHT_SOFTKEY_CODE) {

            // draw inactive menu again

            clearScreen();

            menu.drawInactiveMenu(this, g);

            menuIsActive = false;

        } // end if "Cancel" was pressed on active menu
        // otherwise check navigation

        keyCode = getGameAction(keyCode);

        if (keyCode == UP) {

            currentlySelectedIndex--;

            if (currentlySelectedIndex < 0) {

                currentlySelectedIndex = 0; // stay within limits

            }

            clearScreen();

            menu.drawActiveMenu(this, g, currentlySelectedIndex); // repaint active menu

        } // end if UP button was pressed
        else if (keyCode == DOWN) {

            currentlySelectedIndex++;

            if (currentlySelectedIndex >= menuOptions.length) {

                currentlySelectedIndex = menuOptions.length - 1; // stay within limits

            }

            clearScreen();

            menu.drawActiveMenu(this, g, currentlySelectedIndex); // repaint active menu

        } // end if DOWN button was pressed
        else if (keyCode == FIRE) {

            // menu option is selected
            // simply draw selected option

            clearScreen();

        }

    }

}
```

How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

```
        g.setColor(0x000000); // black

        g.drawString("[ " + menuOptions[currentlySelectedIndex] + " ] was selected", 10, 15

        menu.drawInactiveMenu(this, g);

        menuIsActive = false;

    } // end if FIRE button was pressed

} // end if menu is active
else { // draw inactive menu

    // check if the "Options" or "Exit" buttons were pressed

    if (keyCode == LEFT_SOFTKEY_CODE) { // "Options" pressed

        clearScreen();

        currentlySelectedIndex = 0;

        menu.drawActiveMenu(this, g, currentlySelectedIndex); // activate menu

        menuIsActive = true;

    } // end if "Options" was pressed
    else if (keyCode == RIGHT_SOFTKEY_CODE) {

        exitGUI();

    } // end if "Exit" was pressed

} // end if menu is not active

} // end keyPressed

public void exitGUI() {

    midlet.destroyApp(false);
    midlet.notifyDestroyed();

} // end exitGUI

public void clearScreen() {

    g.setColor(0xffffffff); // white
    g.fillRect(0, 0, width, height);
    flushGraphics();

} // end clearScreen

} // end GUI.java
```

Menu.java

```
import java.util.*;
```

How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

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

public class Menu {

    private String leftOption; // will be displayed when menu is inactive
    private String rightOption; // will be displayed when menu is inactive
    private String cancelOption = "Cancel"; // also may be "Back" or something else
    private String[] menuOptions;

    private int padding = 5; // just like in CSS

    /**
     * Creates a new instance of Menu.
     */
    public Menu(String leftOption, String rightOption, String[] menuOptions) {

        this.leftOption = leftOption;
        this.rightOption = rightOption;
        this.menuOptions = menuOptions;
    } // end constructor

    public void drawInactiveMenu(GameCanvas canvas, Graphics g) {

        // create inactive menu font

        Font font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM);

        int fontHeight = font.getHeight();

        // clear inactive menu background

        int width = canvas.getWidth();
        int height = canvas.getHeight();

        g.setColor(0xcccccc); // grey color
        g.fillRect(0, height - fontHeight - 2 * padding, width, height);

        // draw left and right menu options

        g.setFont(font);
        g.setColor(0x000000); // black

        g.drawString(leftOption, padding, height - padding, g.LEFT | g.BOTTOM);
        g.drawString(rightOption, width - padding, height - padding, g.RIGHT | g.BOTTOM);
        canvas.flushGraphics();
    } // end drawInactiveMenu

    public void drawActiveMenu(GameCanvas canvas, Graphics g, int selectedOptionIndex) {

        // create active menu font

        Font font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_MEDIUM);

        int fontHeight = font.getHeight();
```

How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

```
// clear menu bar background

int width = canvas.getWidth();
int height = canvas.getHeight();

g.setColor(0xcccccc);
g.fillRect(0, height - fontHeight - 2 * padding, width, height);

// draw default menu bar options

g.setFont(font);
g.setColor(0x000000); // black

// draw "Cancel" option
g.drawString(cancelOption, width - padding, height - padding, g.RIGHT | g.BOTTOM);
canvas.flushGraphics();

// draw menu options

if (menuOptions != null) {

    // check out the max width of a menu (for the specified menu font)

    int menuMaxWidth = 0;
    int menuMaxHeight = 0;
    int currentWidth = 0;

    // we'll simply check each option and find the maximal width

    for (int i = 0; i < menuOptions.length; i++) {

        currentWidth = font.stringWidth(menuOptions[i]);

        if (currentWidth > menuMaxWidth) {
            menuMaxWidth = currentWidth; // update
        }

        menuMaxHeight += fontHeight + padding; // for a current menu option
    } // end for each menu option

    menuMaxWidth += 2 * padding; // padding from left and right

    // now we know the bounds of active menu
    // draw active menu's background

    g.setColor(0xcccccc);
    g.fillRect(0, // x
               height - fontHeight - 2 * padding - menuMaxHeight, // y
               menuMaxWidth,
               menuMaxHeight);

    // draw menu options (from up to bottom)

    g.setFont(font);

    int menuOptionX = padding;
    int menuOptionY = height - fontHeight - 2 * padding - menuMaxHeight + padding;

    for (int i = 0; i < menuOptions.length; i++) {

        if (i != selectedOptionIndex) { // draw unselected menu option
```

How_to_create_an_advanced_menu_in_Java_ME_using_the_low-level_graphical_API

```
        g.setColor(0x000000); // black
    } // end if draw unselected menu option
    else { // draw selected menu option

        /**
         * The simplest way to separate selected menu option
         * is by drawing it with different color.
         * However, it also may be painted as underlined text
         * or with different background color.
         */
        g.setColor(0x0000ff); // blue

    } // end if draw selected menu option

    g.drawString(menuOptions[i], menuOptionX, menuOptionY, g.LEFT | g.TOP);

    menuOptionY += padding + fontHeight;

} // end for each menu option

canvas.flushGraphics();

} // end if menu options were specified

} // end drawActiveMenu

} // end Menu.java
```