Handling (releasing) camera resource in Java ME

Overview

This code snippet demonstrates how to handle camera resources when an application starts, pauses, resumes, or terminates.

HandleCameraResource.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.Alert;

import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.control.VideoControl;
import javax.microedition.media.MediaException;

import java.io.IOException;


public class HandleCameraResource extends MIDlet implements CommandListener {

    private Display display;

    // Form where camera viewfinder is placed.
    private Form cameraForm;
    // Command for exiting from midlet. Placed in cameraForm.
    private Command cmdExit;

    // Player for camera.
    private Player player;
    // Video control (viewfinder) of camera.
    private VideoControl videoControl;

    // Alert to be displayed if error occurs.
    private Alert alert;

    /**
     * Constructor.
     */
    public HandleCameraResource() {
        initializeComponents();
    }

    /**
     * Initializes components of midlet.
     */
    private void initializeComponents() {
        display = Display.getDisplay(this);

        if(checkCameraSupport() == false) {
            showAlert("Error", "Camera is not supported!", null);
            return;
        }

        try {
            createCameraForm();
            createCameraPlayer();
            addCameraViewfinderToForm();
        } catch(IOException ioExc) {
            showAlert("IO error", ioExc.getMessage(), null);
        } catch(MediaException mediaExc) {
            showAlert("Media error", mediaExc.getMessage(), null);
        }
    }

    /**
     *  Creates and returns form where the camera control will be placed.
     */
    private void createCameraForm() {
        // Create camera form
        cameraForm = new Form("Camera");
        // Create commands for this form
        cmdExit = new Command("Exit", Command.EXIT, 0);
        // Add commands to form
        cameraForm.addCommand(cmdExit);
        // Set midlet as command listener for this form
        cameraForm.setCommandListener(this);
    }

    /**
     * Check camera support.
     * @return true if camera is supported, false otherwise.
     */
    private boolean checkCameraSupport() {
        String propValue = System.getProperty("supports.video.capture");
        return (propValue != null) && (propValue.equals("true") == true);
    }

    /**
     * Creates camera player and viewfinder.
     * @throws IOException if creation of player is failed.
     * @throws MediaException if creation of player is failed.
     */
    private void createCameraPlayer() throws IOException, MediaException {
       // Create player for camera
     try {
```


(C) Copyright Nokia 2013. All rights reserved.

```
    player = Manager.createPlayer("capture://video");
   }
   catch (MediaException exc) {
    player = Manager.createPlayer("capture://image");
   }
      // Realize player. After realize completes the player is in the REALIZED state.
      player.realize();
      // Acquire resources for player. After prefetch completes
      // the player is in the PREFETCHED state.
      player.prefetch();

      // Create camera viewfinder
      videoControl = (VideoControl)player.getControl("VideoControl");
  }

  /**
   * Adds camera viewfinder as item to cameraForm.
   */
  private void addCameraViewfinderToForm() {
      cameraForm.append((Item)videoControl.
              initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
  }

  /**
   * Starts camera player.
   * Once prefetched, a player can enter the STARTED state by calling
   * the start method. A STARTED player means the Player is running
   * and processing data.
   * @throws MediaException if staring of player is filed.
   */
  private void startCameraPlayer() throws MediaException {
      if(player != null) {
          if(player.getState() == Player.PREFETCHED) {
              player.start();
          }
      }
  }

  /**
   * Stops camera player.
   * When stop method returns, the Player is in the PREFETCHED state.
   */
  private void stopCameraPlayer() {
      if(player != null) {
          try {
              if(player.getState() == Player.STARTED) {
                  player.stop();
              }
          }
          catch(MediaException mediaExc) {
              showAlert("media error", mediaExc.getMessage(), cameraForm);
          }
          catch(IllegalStateException illStateExc) {
              showAlert("Illegal state error", illStateExc.getMessage(),
                      cameraForm);
          }
      }
  }

  /**
   * Stops and destroys camera player.
   */
  private void destroyCameraPlayer() {
      if(player != null) {
          try {
              player.deallocate();
          }
          catch(IllegalStateException illStateExc) {
              showAlert("Illegal state error", illStateExc.getMessage(), null);
          }
      }
  }

  /**
   * Shows alert with specified title and text. If next displayable is not
   * specified then application will be closed after alert closing.
   * @param title - Title of alert.
   * @param message - text of alert.
   * @param nextDisp - next displayable. Can be null.
   */
  private void showAlert(String title, String message, Displayable nextDisp) {
      alert = new Alert(title);
      alert.setString(message);
      alert.setTimeout(Alert.FOREVER);

      if(nextDisp != null) {
          display.setCurrent(alert, nextDisp);
      } else {
          display.setCurrent(alert);
          alert.setCommandListener(this);
      }
  }

  /**
   * From MIDlet.
   * Signals the MIDlet that it has entered the Active state.
   * Set cameraForm as current displayable and run camera player.
   */
  public void startApp() {
      display.setCurrent(cameraForm);
      try {
          startCameraPlayer();
      } catch (MediaException mediaExc) {
          showAlert("Media error", mediaExc.getMessage(), null);
      }
  }

  /**
   * From MIDlet.
   * Signals the MIDlet to enter the Paused state.
   * Stop camera player while application is paused.
   */
  public void pauseApp() {
      stopCameraPlayer();
  }

  /**
   * Performs exit from midlet.
   */
  public void exitMIDlet() {
```


(C) Copyright Nokia 2013. All rights reserved.

```
        notifyDestroyed();
    }

    /**
     * From MIDlet.
     * Signals the MIDlet to terminate and enter the Destroyed state.
     * Stop and destroy camera player.
     */
    public void destroyApp(boolean unconditional) {
        destroyCameraPlayer();
    }

    /**
     * From CommandListener.
     * Indicates that a command event has occurred on Displayable displayable.
     * @param command - a Command object identifying the command.
     * @param displayable - the Displayable on which this event has occurred.
     */
    public void commandAction(Command command, Displayable displayable) {
        // Handles "exit" command
        if(command == cmdExit) {
            exitMIDlet();
        }
        // Handle "ok" command from alert
        if(displayable == alert) {
            exitMIDlet();
        }
        // Handle "ok" command from alert
        if(displayable == alert) {
            exitMIDlet();
        }
    }
}
```

Postconditions

This code snippet represents a midlet that handles camera resources properly.

Supplementary material

Executables and source files can be found in Media:HandlingCameraResources.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.