Synchronising resource access in Java ME

This code snippet demonstrates how to create and execute several threads and synchronise their access to a resource.

Overview

Because a context switch can occur at any time, a new running thread can find itself reading data that a previous running thread has not yet finished writing. Interacting threads must use thread synchronisation in order to control which thread can read or write shared data at any given moment.

Thread synchronisation depends on a construct called a monitor. The monitor acts as a gatekeeper, ensuring that only one thread at a time has access to specific data. Whenever a thread needs to access protected data, it asks the monitor to lock the data for its use. Other threads trying to access the same data are forced to wait until the data is unlocked. After that, the monitor will lock the data for one of the waiting threads and allow it to proceed with its processing.

Any Java object can act as a monitor to control access to data using the `synchronized` keyword.

In this snippet, the resource shared between threads is the `StringContainer` class. It contains a string and the `addLetter()` method, which adds a specified character to the string three times. A synchronised version of this method is named `addLetterSync()`. This method consist of a `synchronized` block, which ensures that only one thread can execute this block updating string at a time. In this case, the current thread adds exactly three specified characters to the string before any other thread can write to the string. To switch between executing the synchronised and the non-synchronised version of `addLetter`, the user can press the 'Synchronizing' check box on the main form. When `addLetterSync()` is executing, different letters appear in the text field exactly three times. When `addLetter` is executing, different letters appear in the text field in random order.

In this snippet, three threads write their characters to the string - which is located in the StringContainer class - and one other thread returns this string into a text field placed on a form. All these thread operations are performed in synchronised or non-synchronised way depending on the 'Synchronizing' check box.

This MIDlet consists of 4 source files:

1. SynchronizingMidlet.java - contains the MIDlet class.
2. StringContainer.java - contains the resource class shared between threads.
3. AddCharThread.java - contains the thread class for updating the string container.
4. OutputStringThread.java - contains the thread class for showing the string container state to the user.

Source file: SynchronizingMidlet.java

```
    // Text field contains output from string container
    private TextField textField;

    // Access synchronization choise group
    private ChoiceGroup syncTypeChoise;

    // String container to be updated by threads
    private StringContainer stringContainer;
    // Array of threads which updates string container
    private Vector threads;
    // Thread which returns and stores state of string container
    private OutputStringThread outputThread;

    /**
     * Sets up the main form.
     */
    private void setupMainForm() {
        mainForm = new Form("Synchronizing resource access");
```


(C) Copyright Nokia 2014. All rights reserved.

```
        ...

        // Create list box for synchronized type
        syncTypeChoise = new ChoiceGroup("",
                ChoiceGroup.MULTIPLE);
        syncTypeChoise.append("synchronize", null);
        syncTypeChoise.setSelectedIndex(0, true);

        mainForm.append(syncTypeChoise);

        mainForm.setItemStateListener(this);

        ...
    }

    /**
     * Executes the snippet.
     */
    private void executeSnippet() {
        stringContainer = new StringContainer(15);
        startThreads();
    }

    /**
     * Closes the opened resources and exits the application.
     */
    private void exit() {
        stopThreads();

        ...

        notifyDestroyed();
    }


    /**
     * Starts threads for updating string container and sending it state
     * to text field.
     */
    private void startThreads() {
        // Stop threads if they are already running
        stopThreads();

        // Get new value of sync ops flag
        boolean useSyncOps = syncTypeChoise.isSelected(0);

        // Create threads for updating string container. Each thread will write
        // to string container one of chars from array below three times.
        char[] chars = {'a', 'b', 'c'};
        threads = new Vector(chars.length);
        for(int n = 0; n < chars.length; n++) {
            AddCharThread thread = new AddCharThread(stringContainer, chars[n]);
            thread.setUseSyncOperations(useSyncOps);
            threads.addElement(thread);
            thread.start();
        }
```


(C) Copyright Nokia 2014. All rights reserved.

```
        // Create thread for output
        outputThread = new OutputStringThread(textField, stringContainer);
        outputThread.start();
    }

    /**
     * Stops all threads.
     */
    private void stopThreads() {
        // Stop update threads
        if(threads != null) {
            for(int n = 0; n < threads.size(); n++) {
                AddCharThread thread = (AddCharThread)threads.elementAt(n);
                if(thread != null && thread.isAlive() == true) {
                    thread.quit();
                    try {
                        thread.join();
                    } catch(InterruptedException exc) {
                        // This thread was interrupted, do nothing.
                    }
                }
            }

            threads.removeAllElements();
        }

        // Stop output thread
        if(outputThread != null && outputThread.isAlive() == true) {
            outputThread.quit();
            try {
                outputThread.join();
            } catch(InterruptedException exc) {
                // This thread was interrupted, do nothing.
            }
        }
    }

    /**
     * From ItemStateListener.
     * Called when internal state of an Item has been changed by the user.
     * @param item the item that was changed
     */
    public void itemStateChanged(Item item) {
        if(item == syncTypeChoise) {
            // Get new value of flag
            boolean useSyncOps = syncTypeChoise.isSelected(0);

            // Set new value to all char threads
            int threadCount = threads.size();
            for(int n = 0; n < threadCount; n++) {
                AddCharThread thread = (AddCharThread)threads.elementAt(n);
                thread.setUseSyncOperations(useSyncOps);
            }
        }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

Source file: StringContainer.java

```
/**
 * This class contains string and some synchronized operations for it.
 */
public class StringContainer {
    // String buffer contains string
    private StringBuffer stringBuffer;
    // Max length of the string
    private int maxLength;

    /**
     * Constructor
     * @param maxLength - max length of the string in container.
     */
    public StringContainer(int maxLength) {
        this.maxLength = maxLength;
        stringBuffer = new StringBuffer();
    }

    /**
     * Adds specified character to the end of string three times.
     * If length of string exceeding max length then characters from begin
     * of string will be removed.
     * This method is not syncronized.
     * @param ch - character for updating string.
     */
    public void addLetter(char ch) {
        // Add character to buffer
        for(int n = 0; n < 3; n++) {
            stringBuffer.append(ch);
            // If buffer length exceed max length then cut it
            if(stringBuffer.length() > maxLength) {
                stringBuffer.deleteCharAt(0);
            }
        }
    }

    /**
     * Adds specified character to the end of string three times.
     * This method is a syncronized version of addLetter method.
     * @param ch - character for updating string.
     */
    public void addLetterSync(char ch) {
        synchronized(stringBuffer) {
            addLetter(ch);
        }
    }

    /**
     * Returns string from this container. This method is not synchronized.
     * @return string from this container.
     */
    public String toString() {
```


(C) Copyright Nokia 2014. All rights reserved.

```
        return stringBuffer.toString();
    }

    /**
     * Returns string from this container. This method is synchronized version
     * of toString method.
     * @return string from this container.
     */
    public String toStringSync() {
        synchronized(stringBuffer) {
            return toString();
        }
    }
}
```

Source file: AddCharThread.java

```
/**
 * AddCharThread class.
 * Adds character to specified string container.
 */
public class AddCharThread extends Thread {
    // String container to be updated
    private StringContainer stringContainer;
    // Character for updating string container
    private char ch;
    // Flag indicating that thread must quit.
    private boolean isQuit = false;

    // Flag indicating using synchronized operations of stringContainer or not.
    // Type of this class member is Boolean instead of boolean. It is done for
    // synchronized purposes since only reference types can be synchronized.
    private Boolean bUseSyncOperations = Boolean.TRUE;

    /**
     * Constructor.
     * @param stringContainer - resource to be updated
     * @param ch - character for updating string container
     */
    public AddCharThread(StringContainer stringContainer, char ch) {
        this.stringContainer = stringContainer;
        this.ch = ch;
    }

    /**
     * Sets flag indicating using synchronized operations of stringContainer
     * or not.
     * @param bUse if true - use synchronized operations.
     */
    public void setUseSyncOperations(boolean bUse) {
        synchronized(bUseSyncOperations) {
            bUseSyncOperations = new Boolean(bUse);
        }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    /**
     * Gets flag indicating using synchronized operations of stringContainer
     * or not.
     * @return
     */
    private boolean getUseSyncOperations() {
        synchronized(bUseSyncOperations) {
            return bUseSyncOperations.booleanValue();
        }
    }
```

Source file: OutputStringThread.java

```
import javax.microedition.lcdui.TextField;

/**
 * OutputStringThread class
 * Output current state of specified string container to the text field.
 */
public class OutputStringThread extends Thread {
    // Text field for storing output string container state
    private TextField textField;
    // String container to store its state
    private StringContainer stringContainer;
    // Indicates if this thread must be terminated
    private boolean isQuit = false;

    /**
     * Constructor.
     * @param textField - text field for storing output string container state
     * @param container - string container for storing its state
     */
    public OutputStringThread(TextField textField, StringContainer container) {
        this.textField = textField;
        this.stringContainer = container;
    }

    /**
     * Terminates thread.
     */
    public void quit() {
        isQuit = true;
    }

    /**
     * From Thread class.
     * Implements thread functionality.
     */
    public void run() {
        // If something is not assigned then exit thread
        if(textField == null || stringContainer == null) {
            return;
        }
```


(C) Copyright Nokia 2014. All rights reserved.

```
        // Thread main loop
        while(isQuit == false) {
            // Store into text field state of string container
            textField.setString(stringContainer.toStringSync());
            // Sleep for a 200 milliseconds
            try {
                sleep(200);
            } catch(InterruptedException exc) {
                // Thread was interrupted
                return;
            }
        }
    }
}
```

Postconditions

A text field, the 'Synchronizing' check box and a logging text box are placed on the display. Press 'Execute snippet' to start the threads which write their characters to the string in a synchronised way, and one other thread which returns this string in the text field placed on the form.

Version Hint

Windows Phone: [[Category:Windows Phone]]
[[Category:Windows Phone 7.5]]
[[Category:Windows Phone 8]]

Nokia Asha: [[Category:Nokia Asha]]
[[Category:Nokia Asha Platform 1.0]]

Series 40: [[Category:Series 40]]
[[Category:Series 40 1st Edition]] [[Category:Series 40 2nd Edition]]
[[Category:Series 40 3rd Edition (initial release)]] [[Category:Series 40 3rd Edition FP1]] [[Category:Series 40 3rd Edition FP2]]
[[Category:Series 40 5th Edition (initial release)]] [[Category:Series 40 5th Edition FP1]]
[[Category:Series 40 6th Edition (initial release)]] [[Category:Series 40 6th Edition FP1]] [[Category:Series 40 Developer Platform 1.0]] [[Category:Series 40 Developer Platform 1.1]] [[Category:Series 40 Developer Platform 2.0]]

Symbian: [[Category:Symbian]]
[[Category:S60 1st Edition]] [[Category:S60 2nd Edition (initial release)]] [[Category:S60 2nd Edition FP1]] [[Category:S60 2nd Edition FP2]] [[Category:S60 2nd Edition FP3]]
[[Category:S60 3rd Edition (initial release)]] [[Category:S60 3rd Edition FP1]] [[Category:S60 3rd Edition FP2]]
[[Category:S60 5th Edition]]
[[Category:Symbian^3]] [[Category:Symbian Anna]] [[Category:Nokia Belle]]

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.