Using List in Java ME

The code example demonstrates how to use several standard types of LCDUI listboxes.

Overview

There are four types of lists in MIDP2.0 / LCDUI:

```
Choice.IMPLICIT
Choice.EXCLUSIVE
Choice.MULTIPLE
Choice.POPUP
```

`List` implements the `Choice` interface. Only first three can be used for List objects.

`IMPLICIT` - A list of strings. Each item gets selected when a cursor moves over it. `List.setSelectCommand` can be used with this type of List. You can redefine a Command that will be triggered on selection. Selected item index can be retrieved using the `List.getSelectedIndex` method.

`EXCLUSIVE` A choice that has exactly one element selected at time. An item gets selected only when you move the cursor over it and press 'Select'. The selected item index can be retrieved using the `List.getSelectedIndex` method.

`MULTIPLE` - A list of checkboxes. This type of list provides multiple selection functionality. Selected items can be retrieved using the `List.getSelectedFlag` method.

Source file: ListBoxDemo.java

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;


public class ListBoxDemo extends MIDlet implements CommandListener
{
    private Display display;
    /**
     * IMPLICIT listbox.
     */
    private List lbxImpl;
    /**
     * EXPLICIT listbox.
     */
    private List lbxExcl;
    /**
     * MULTIPLE listbox.
     */
    private List lbxMult;
    /**
     * Item count.
     */
    private int arrItemCount;
    /**
     * Selection command for IMPLICIT listbox.
     */
    private Command cmdSel;
    /**
     * Command that gets currently selected items.
     */
    private Command cmdSelected;
    /**
     * Calls exitMIDlet method.
     */
    private Command cmdExit;
    /**
     * Shows IMPLICIT listbox.
     */
    private Command cmdImpl;
    /**
     * Shows EXPLICIT listbox.
     */
    private Command cmdExcl;
    /**
     * Shows MULTIPLE listbox.
     */
    private Command cmdMult;
    /**
     * Constructor. Constructs the object and initializes displayables.
     */
    public ListBoxDemo() {
        display = Display.getDisplay( this );
        arrItemCount = 5;
        String[] arrItems = new String[arrItemCount];
        for(int i = 0; i < arrItemCount; i++) {
            arrItems[i] = "Item " + i;
        }

        cmdExit = new Command( "Exit", Command.EXIT, 1 );
        cmdImpl = new Command( "Implicit", Command.SCREEN, 1);
        cmdExcl = new Command( "Exclusive", Command.SCREEN, 1);
        cmdMult = new Command( "Multiple", Command.SCREEN, 1);
```


(C) Copyright Nokia 2013. All rights reserved.

```
        cmdSelected = new Command( "Show selected", Command.SCREEN, 1 );
        cmdSel = new Command( "Select", Command.ITEM, 1 );
        lbxImpl = createListBox( List.IMPLICIT, arrItems );
        lbxExcl = createListBox( List.EXCLUSIVE, arrItems );
        lbxMult = createListBox( List.MULTIPLE, arrItems );
    }
    /**
     * Creates a List object instance, fills it with data
     * and adds commands to it.
     * @param lbxType - holds type of listbox to create.
     * @param arrItems - array of string items to fill listbox with.
     * @return Reference to created List object instance.
     */
    private List createListBox( int lbxType, String[] arrItems ) {
        List lbxTemp = null;
        lbxTemp = new List( "Listbox",
               lbxType, //setting list box type
               arrItems, //setting listbox content
               null);  //we don't set any images
        //setting commands
        lbxTemp.addCommand( cmdExit );
        lbxTemp.addCommand( cmdImpl );
        lbxTemp.addCommand( cmdExcl  );
        lbxTemp.addCommand( cmdMult );
        lbxTemp.addCommand( cmdSelected );
        if( lbxType == List.IMPLICIT ) {
            lbxTemp.setSelectCommand( cmdSel );
        }
        lbxTemp.setCommandListener( this );
        return lbxTemp;
    }
    /**
     * Returns a String hodling " " separater selected items
     * indexes.
     */
    private String getSelectedItems( List lbx ) {
        boolean[] arrSel = new boolean[lbx.size()];
        int selected = lbx.getSelectedFlags( arrSel );
        String result = "";
        for(int i = 0; i< arrSel.length;i++ ) {
            if( arrSel[i] ) {
                result = result + " " + i;
            }
        }
        return result;
    }
    /**
     * From MIDlet.
     * Called when MIDlet is started.
     * @throws javax.microedition.midlet.MIDletStateChangeException
     */
    public void startApp() throws MIDletStateChangeException {
        display.setCurrent( lbxImpl  );
    }
    /**
     * From MIDlet.
     * Called to signal the MIDlet to enter the Paused state.
     */
    public void pauseApp() {
        //No implementation required
    }
    /**
     * From MIDlet.
     * Called to signal the MIDlet to terminate.
     * @param unconditional whether the MIDlet has to be unconditionally
     * terminated
     * @throws javax.microedition.midlet.MIDletStateChangeException
     */
    public void destroyApp(boolean unconditional)
        throws MIDletStateChangeException {
    }
    /**
     * From CommandListener.
     * Called by the system to indicate that a command has been invoked on a
     * particular displayable.
     * @param command the command that was invoked
     * @param displayable the displayable where the command was invoked
     */
    public void commandAction( Command command, Displayable displayable ) {

        if( command == cmdExit ) {
            notifyDestroyed();
        }
        if( command == cmdSel ) {
            lbxImpl.setTitle("Selected: " + lbxImpl.getSelectedIndex() );
        }

        if( command == cmdSelected ) {
            if( displayable == lbxImpl ){
                lbxImpl.setTitle("Selected: " +
                            lbxImpl.getSelectedIndex() );
            }
            if( displayable == lbxExcl ){
                lbxExcl.setTitle("Selected: " +
                            lbxExcl.getSelectedIndex() );
            }
            if( displayable == lbxMult ){
                lbxMult.setTitle("Selected:" + getSelectedItems(lbxMult) );
            }
        }
        if( command == cmdImpl ) {
            display.setCurrent( lbxImpl  );
        }
        if( command == cmdExcl ) {
            display.setCurrent( lbxExcl  );
        }
        if( command == cmdMult ) {
            display.setCurrent( lbxMult  );
        }
    }
}
```

Postconditions

When the application starts, an `IMPLICIT` listbox is being opened.

(C) Copyright Nokia 2013. All rights reserved.

'Show selected' command displays currently selected item/items in the title bar.

For `IMPLICIT` listbox, when you press 'Select', the title is immidiately changed.

For `EXCLUSIVE` listbox items, selection status changes only when you press 'Select'.

`MULTIPLE` listbox have check/uncheck commands instead of 'Select'.

After checking the specific items, their indexes can be retrieved by using `List.getSelectedFlags`.

Supplementary material

You can view the source file and executable application in the attached zip archive. The archive is available for download at Media:Using ListBox in Java ME.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.