Event-driven programming with Java leJOS

Versión 0.1

Juan Antonio Breña Moral

7-jul-08

Index

1.- Introduction 4

1.1.- Goals 4

1.1.1.- About this document 4

1.2.- LeJOS Project 4

1.3.- NXT Brick 5

1.3.1.- NXT Sensors used in the eBook 6

1.4.- About the author 7

2.- Event-driven programming 8

2.1.- Introduction 8

2.2.- General concepts 8

2.3.- Example with Java AWT 9

2.3.1.- Example1: EventTest.java 9

2.3.2.- Example2: EventTest.java 11

2.4.- Events with Java leJOS 13

2.5.- Examples with Java leJOS 13

2.5.1.- Example1: ListenForButtons 13

2.5.2.- Example2: KeyboardTest 14

2.6.- Links 17

Revision History

Name	Date	Reason For Changes	Version
Juan Antonio Breña Moral	05/07/2008		0.1

1.- Introduction

1.1.- Goals

Many developers around the world choose leJOS, Java for Lego Mindstorm, as the main platform to develop robots with NXT Lego Mindstorm. I consider that this eBook will help leJOS community, Lego Mindstorm community, Robot's developers and Java fans to develop better software.

Robotics will be very important for the humanity in the next 10 years and this eBook is an effort to help in this way.

Many people spend several hours in their robotics projects with problems with wires & electronics, protocols and problems with programming languages, Lego Mindstorm is easy and Java/leJOS is an excellent platform to demonstrate your software engineering skills to develop better robots. NXT Brick is the easiest way to enter in the robotics world and leJOS, the best platform in the moment to use software engineering ideas.

Enjoy, Learn, Contact with me to improve the eBook and share your ideas.

Juan Antonio Breña Moral.
www.juanantonio.info

1.1.1.- About this document

This document will explain Event-driven programming with Java and how to use events with NXJ.

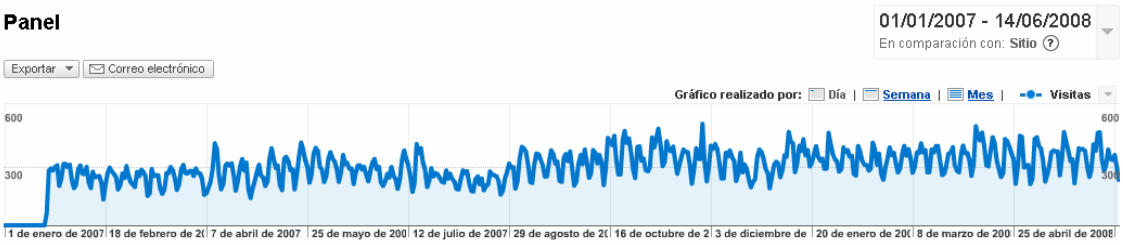
1.2.- LeJOS Project

LeJOS is Sourceforge project created to develop a technological infrastructure to develop software into Lego Mindstorm Products using Java technology.

Currently leJOS has opened the following research lines:

1. NXT Technology
    a. NXJ
    b. LeJOS PC API
    c. iCommand
2. RCX Technology
    a. leJOS for RCX

LeJOS project's audience has increased. Currently more than 500 people visit the website every day.



This eBook will focus in NXT technology with NXJ using a Windows Environment to develop software.

1.3.- NXT Brick

The NXT is the brain of a MINDSTORMS robot. It's an intelligent, computer-controlled LEGO brick that lets a MINDSTORMS robot come alive and perform different operations.

Motor ports
The NXT has three output ports for attaching motors - Ports A, B and C

Sensor ports
The NXT has four input ports for attaching sensors - Ports 1, 2, 3 and 4.

USB port
Connect a USB cable to the USB port and download programs from your computer to the NXT (or upload data from the robot to your computer). You can also use the wireless Bluetooth connection for uploading and downloading.

Loudspeaker
Make a program with real sounds and listen to them when you run the program

NXT Buttons
Orange button: On/Enter /Run
Light grey arrows: Used for moving left and right in the NXT menu

Dark grey button: Clear/Go back

NXT Display
Your NXT comes with many display features - see the MINDSTORMS NXT Users Guide that comes with your NXT kit for specific information on display icons and options

Technical specifications

- 32-bit ARM7 microcontroller
- 256 Kbytes FLASH, 64 Kbytes RAM
- 8-bit AVR microcontroller
- 4 Kbytes FLASH, 512 Byte RAM
- Bluetooth wireless communication (Bluetooth Class II V2.0 compliant)
- USB full speed port
- 4 input ports, 6-wire cable digital platform (One port includes a IEC 61158 Type 4/EN 50 170 compliant expansion port for future use)
- 3 output ports, 6-wire cable digital platform
- 100 x 64 pixel LCD graphical display
- Loudspeaker - 8 kHz sound quality. Sound channel with 8-bit resolution and 2-16 KHz sample rate.
- Power source: 6 AA batteries

1.3.1.- NXT Sensors used in the eBook

NXT Sensors used in the document are the following:

- NXT Motor
- Ultrasonic Sensor
- Compass Sensor
- NXTCam
- Tilt Sensor
- NXTCam
- NXTe

NXT Motor

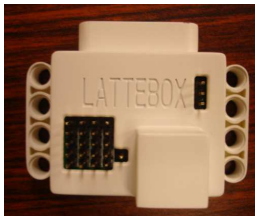
Ultrasonic Sensor

Compass Sensor

Tilt Sensor

NXTCam

NXTe



1.4.- About the author

Juan Antonio Breña Moral has collaborated in leJOS Research team since 2006. He works in Europe leading Marketing, Engineering and IT projects for middle and large customers in several markets as Defence, Telecommunications, Pharmaceutics, Energy, Automobile, Construction, Insurance and Internet.

Further information:
www.juanantonio.info
www.esmeta.es

2.- Event-driven programming

2.1.- Introduction

In Java, any console application has a lineal execution which was programmed in the method **main**. When you develop any application with a Graphical User Interface, GUI, you can use event-driving programming.

- Batch-oriented processing
  - The control of execution is within the program
  - The program reads more information when it needs it, and prints out results when they are ready
- Event-driven GUI processing
  - Processing happens as reactions to the interactive manipulation of visual components

2.2.- General concepts

In Java the concepts related to Events are:

- Event Sources
- Event Listeners

An event source is a Button or a Keyboard por example.
An eventListener is a class interested in handling certain kinds of events. A listener must implement an appropriate listener interface then it informs the source that is is interested in handling a certain type of event (registration process). Event code is is executed when events are generated by user actions, such as:

- Click a button
- Keystrokes

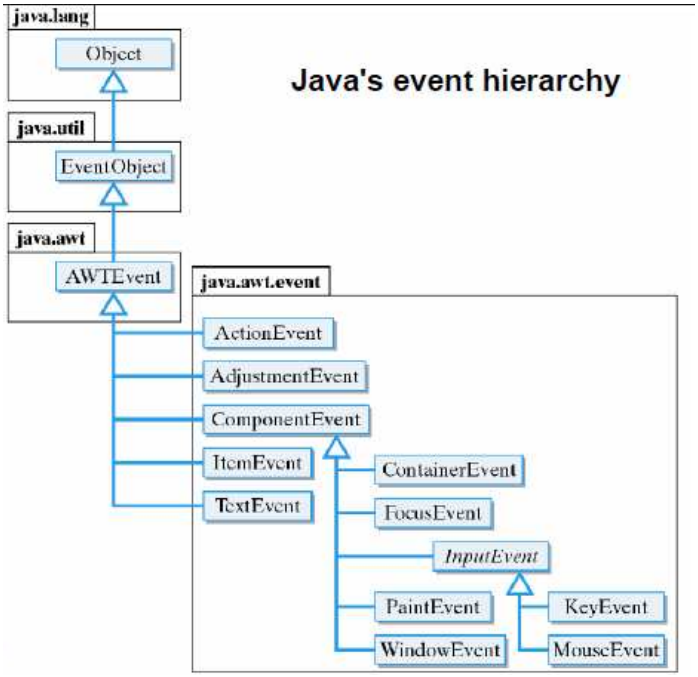
The class EventObject is the base class used to handle every event in Java. In Java the event heriachy is the following:

Event notification
The notification mechanism involves calling a specific method of any objects that want to be notified about the event and passing the event object as a parameter. These methods are called *event handlers*. For an object to be notified about an event, it must implement a specific interface and be *registered* with the source of the event. These objects are called *event listeners*.

The interface the object must implement depends on the event it wants to receive. There are interfaces for each kind of event. All of them are subinterfaces of the java.util.EventListener interface. Each interface defines one or more methods that are called by the source of the event. Each method receives an event object as a parameter.

For example, a javax.swing.JButton object generates a java.awt.event.ActionEvent when pressed. It creates an instance of the java.awt.event.ActionEvent class, populates it with information about the event, including a reference to itself, and notifies the interested parties about the event.

The following image shows Java's event Hierarchy:



2.3.- Example with Java AWT

To undertand the concepts, see the following examples.

2.3.1.- Example1: EventTest.java

This example creates a new window with 3 buttons. The example handles the click event and change window's background.

The code:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class EventTest{
	public static void main(String[] args){
		JFrame frame = new EventFrame();
		frame.setVisible(true);
	}
}

class EventFrame extends JFrame{
	public EventFrame(){
```

```
            setTitle("Demo");
            setSize(300,100);
            addWindowListener(new MainWindowListener());

            Container contenido = getContentPane();
            contenido.add(new ButtonPanel());
      }
}

class MainWindowListener extends WindowAdapter{
      public void windowClosing(WindowEvent e){
            System.exit(0);
      }
}

class ButtonPanel extends JPanel implements ActionListener{
      private JButton redButton;
      private JButton greenButton;
      private JButton blueButton;

      public ButtonPanel(){
            redButton = new JButton("Red");
            greenButton = new JButton("Green");
            blueButton = new JButton("Blue");

            this.add(redButton);
            this.add(greenButton);
            this.add(blueButton);

            redButton.addActionListener(this);
            greenButton.addActionListener(this);
            blueButton.addActionListener(this);
      }

      public void actionPerformed (ActionEvent event){
            Object source = event.getSource();
            Color color = getBackground();

            if(source == redButton){
                  color = Color.RED;
            }else if(source == greenButton){
                  color = Color.GREEN;
            }else if(source == blueButton){
                  color = Color.BLUE;
            }

            setBackground(color);
            repaint();
      }
}
```

If you notice the code, the file EventTest has 3 inner classes. The class EventFrame, generates the window and it has registered an event using:

```
addWindowListener(new MainWindowListener());
```

MainWindowListener is class which manages all events about JFrame Objects.

```
class MainWindowListener extends WindowAdapter{
      public void windowClosing(WindowEvent e){
            System.exit(0);
```

```
	}
}
```

If you observe the code, the class ButtonPanel which manages the buttons has implemented the interface ActionListener which was used to manage click event on the buttons.

The way to register the events is:

```
redButton.addActionListener(this);
greenButton.addActionListener(this);
blueButton.addActionListener(this);
```

In this case the method ActionPerformed is used by all buttons.

```
public void actionPerformed (ActionEvent event){
	Object source = event.getSource();
	Color color = getBackground();

	if(source == redButton){
		color = Color.RED;
	}else if(source == greenButton){
		color = Color.GREEN;
	}else if(source == blueButton){
		color = Color.BLUE;
	}

	setBackground(color);
	repaint();
}
```

Every object which manages a sensor keep an internal list of "listener". Listeners are notified (that is, the listener's methods are called) when the user-interface object generates an event. To add listeners to the list you make a call like yellowButton.addActionListener(...). What you put in the place of the ... must be an object implementing the ActionListener interface, so it will have methods capable of processing the events generated. Of course, if the interface is not a button or menu, ActionListener may not be the appropriate interface—there are eleven listener interfaces.

2.3.2.- Example2: EventTest.java

This example creates a new window with a Button and a Textbox. The example handles the click event and manage a internal counter to show the number of clicks.


```
import java.awt.*;
import java.awt.event.*;
```

```
public class AL extends Frame implements WindowListener,ActionListener
{
	TextField text = new TextField(20);
	Button b;
	private int numClicks = 0;

	public static void main(String[] args) {
		AL myWindow = new AL("My first window");
		myWindow.setSize(350,100);
		myWindow.setVisible(true);
	}

	public AL(String title) {
		super(title);
		setLayout(new FlowLayout());
		addWindowListener(this);
		b = new Button("Click me");
		add(b);
		add(text);
		b.addActionListener(this);
	}

	public void actionPerformed(ActionEvent e) {
		numClicks++;
		text.setText("Button Clicked " + numClicks + " times");
	}

	public void windowClosing(WindowEvent e) {
		dispose();
		System.exit(0);
	}

	public void windowOpened(WindowEvent e) {}
	public void windowActivated(WindowEvent e) {}
	public void windowIconified(WindowEvent e) {}
	public void windowDeiconified(WindowEvent e) {}
	public void windowDeactivated(WindowEvent e) {}
	public void windowClosed(WindowEvent e) {}

}
```

In this example, there is a unique class implementing 2 interfaces: WindowListener and Action Listener. If you remember ActionListener has a unique method: ActionPerformed but WindowListener has several methods. When you use Interface with Java, it is necessary to implement all methods. In this case, several methods are declared but without any code inside.

The way to use events is equal to previous example. Register the events:

```
addWindowListener(this);
b.addActionListener(this);
```

And implement the interfaces with the methods:

ActionListener:

```
public void actionPerformed(ActionEvent e) {
	numClicks++;
	text.setText("Button Clicked " + numClicks + " times");
}
```

WindowAdapter:

```
public void windowClosing(WindowEvent e) {
      dispose();
      System.exit(0);
}

public void windowOpened(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
```

2.4.- Events with Java leJOS

Java NXJ supports events. You can use use events with the following Objects:

- Buttons
- Bluetooth keyboard
- SensorPorts

The available EventListeners in NXJ are:

- ButtonListener
- SensorPortListener
- KeyListener

2.5.- Examples with Java leJOS

I have recopilated some examples with Events with leJOS.

2.5.1.- Example1: ListenForButtons

This example registers a ButtonListener in the same method.

```
import  lejos.nxt.*;

public class ButtonEventTest
{
      public static void main (String[] args)
      {
            Button.ENTER.addButtonListener(new myButtonListener());
            while(true);
      }
}

class myButtonListener implements ButtonListener{
      public void buttonPressed(Button b) {
            LCD.drawString("ENTER pressed",0,0);
      }

      public void buttonReleased(Button b) {
            LCD.clear();
      }
}
```

In this example, a GUI in leJOS NXJ register a new Event using a ButtonListener implementation.

To register an EventListener:

```
Button.ENTER.addButtonListener(new myButtonListener());
```

To manage the event:

```
class myButtonListener implements ButtonListener{
	public void buttonPressed(Button b) {
		LCD.drawString("ENTER pressed",0,0);
	}

	public void buttonReleased(Button b) {
		LCD.clear();
	}
}
```

2.5.2.- Example2: KeyboardTest

This example show how implement the Listener in the same class

```
import lejos.nxt.*;
import lejos.nxt.comm.Bluetooth;
import lejos.nxt.comm.RConsole;
import lejos.nxt.comm.NXTConnection;

import javax.bluetooth.RemoteDevice;
import lejos.nxt.comm.BTConnection;
import lejos.devices.*;

import java.util.Vector;
import java.io.*;

/**
 * This is some sample code to demonstrate the Keyboard class. It
 * allows you to connect and display typing on the NXT LCD.
 * Only works with SPP Bluetooth keyboards (very rare). Will not work
with
 * HID BT keyboards. See Keyboard Javadocs for more information.
 * @author BB
 */
public class KeyboardTest implements KeyListener {

	static boolean cont = true;

	public KeyboardTest() {
		Keyboard k = connectKeyboard();
		k.addKeyListener(this);
	}

	public static void main(String [] args) {
		PrintStream(RConsole.openOutputStream()));

		KeyboardTest kt = new KeyboardTest();
		LCD.clear();
		LCD.refresh();
```

```
            while(cont) {Thread.yield();}
            System.out.println("Quitting...");
      }

      public void keyPressed(KeyEvent e) {
            System.out.print("" + e.getKeyChar());
            if(e.getKeyChar() == 'q') {
                  cont = false; // or System.exit(0);
            } else if(e.getKeyChar() == 'z') {
                  System.out.println("");
            }
      }

      public void keyReleased(KeyEvent e) {
            System.out.println("Key Released: " + e.getKeyChar());
            if(e.getKeyChar() == 'q') {
                  cont = false;
            }
      }

      public void keyTyped(KeyEvent e) {
            System.out.println("Key Typed: " + e.getKeyChar());
            if(e.getKeyChar() == 'q') {
                  cont = false;
            }
      }

      public Keyboard connectKeyboard() {

            Keyboard k = null;

            byte[] cod = {0,0,0,0}; // 0,0,0,0 picks up every Bluetooth
device regardless of Class of Device (cod).

            final byte[] pin = {(byte) '0', (byte) '0', (byte) '0',
(byte) '0'};

            InputStream in = null;
            OutputStream out = null;

            System.out.println("Searching ...");
            Vector devList = Bluetooth.inquire(5, 10,cod);

            if (devList.size() > 0) {
                  String[] names = new String[devList.size()];
                  for (int i = 0; i < devList.size(); i++) {
                        RemoteDevice     btrd     =     ((RemoteDevice)
devList.elementAt(i));
                        names[i] = btrd.getFriendlyName(false);
                  }

                  TextMenu searchMenu = new TextMenu(names,1);
                  String[] subItems = {"Connect"};
                  TextMenu subMenu = new TextMenu(subItems,4);

                  int selected;
                  do {
                  LCD.clear();
                        LCD.drawString("Found",6,0);
                        LCD.refresh();
                        selected = searchMenu.select();
```

```
                        if (selected >=0) {
                              RemoteDevice   btrd   =   ((RemoteDevice)
devList.elementAt(selected));
                              LCD.drawString(names[selected],0,1);

      LCD.drawString(btrd.getBluetoothAddress(), 0, 2);
                              int subSelection = subMenu.select();
                              if       (subSelection       ==       0)
Bluetooth.addDevice(btrd);

                              // BELOW: Once paired, no need to use
pin? Might
                              // use alternate method.
                              BTConnection btSPPDevice = null;

                              // Open connection in raw/stream mode
                              btSPPDevice                              =
                        Bluetooth.connect(btrd.getDeviceAddr(),
                        NXTConnection.RAW, pin);

                              try{
                                    in = btSPPDevice.openInputStream();
                                    out                               =
btSPPDevice.openOutputStream();
                                    //System.err.println("Streams
captured.\n");
                                    k = new Keyboard(in, out);
                                    selected = -1; // to make it stop
looping?

                              } catch(Exception e) {
                                    //System.err.println("InputStream
NOT captured." + "\n");
                              }
                        }
                  } while (selected >= 0);

            } else {
                  LCD.clear();
                  LCD.drawString("no devices", 0, 0);
                  LCD.refresh();
                  try {
                        Thread.sleep(2000);
                  } catch (InterruptedException e) {}
            }
            return k;
      }
}
```

If observe the code, the class implement the interface KeyListener which manage the events for BT Keyboard. The interface KeyListener has 3 methods to implement:

- KeyTyped
- KeyPressed
- KeyReleased

Then the class has to implement to run it.

To register the event handler:

```
k.addKeyListener(this);
```

The methods which implement the interface KeyListener:

```
public void keyPressed(KeyEvent e) {
      System.out.print("" + e.getKeyChar());
      if(e.getKeyChar() == 'q') {
            cont = false; // or System.exit(0);
      } else if(e.getKeyChar() == 'z') {
            System.out.println("");
      }
}

public void keyReleased(KeyEvent e) {
      System.out.println("Key Released: " + e.getKeyChar());
      if(e.getKeyChar() == 'q') {
            cont = false;
      }
}

public void keyTyped(KeyEvent e) {
      System.out.println("Key Typed: " + e.getKeyChar());
      if(e.getKeyChar() == 'q') {
            cont = false;
      }
}
```

2.6.- Links

http://www.lejos.org
http://lejos.svn.sourceforge.net/viewvc/lejos/
https://openjdk.dev.java.net/source/browse/openjdk/jdk/trunk/jdk/src/share/classes/java/awt/event/
http://download.java.net/openjdk/jdk7/