Multithreading with Java leJOS

Versión 0.1

Juan Antonio Breña Moral

19-oct-08

Index

Revision History

Name	Date	Reason For Changes	Version
Juan Antonio Breña Moral	27/09/2008		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 in detail how to use Multithreading in Java leJOS to develop a robot which executes task in parallels.

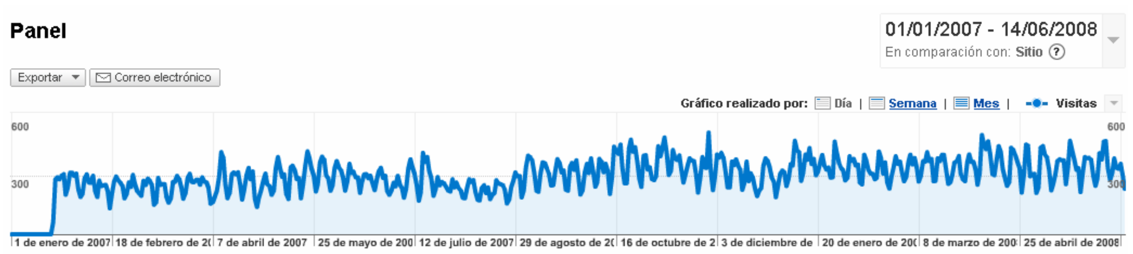
1.2.- LeJOS Project

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. J2ME Mobile API
    d. iCommand
2. RCX Technology
    a. leJOS for RCX

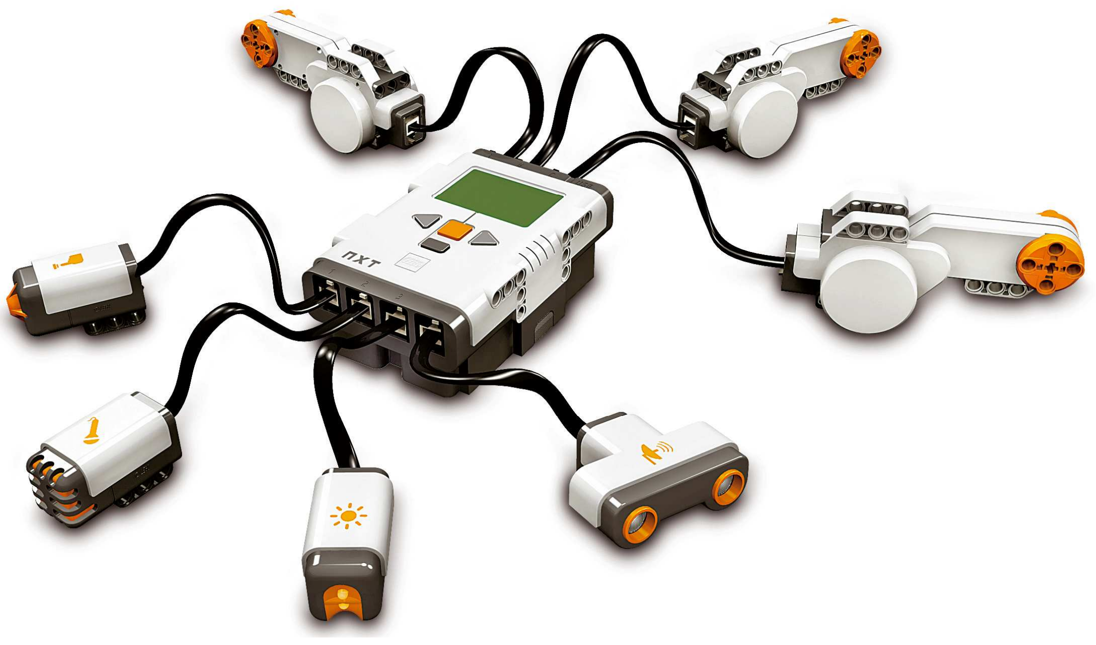
LeJOS project's audience has increased. Currently more than 500 people visit the website every day.



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

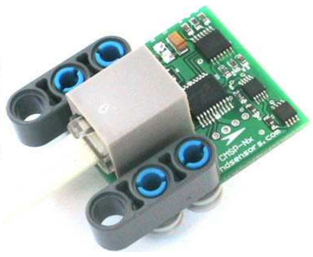
NXT Sensors used in the document are the following:

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

NXT Motor

Ultrasonic Sensor

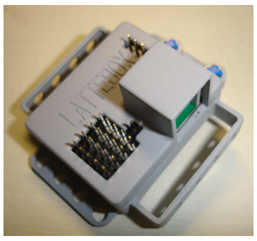
Compass Sensor



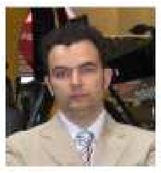
Tilt Sensor

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.- Multithreading with Java leJOS

2.1.- Introduction

Multithreading is a Java feature which allow executing multiple tasks at the same time. If you develop robots, you should consider this programming feature as the basis of your architecture.

Normally when you begin to develop with Java and leJOS you develop programs which execute the operations in sequence:

```
Program:
        Task1
        Task2
        Task3
        ...
        Taskn
```

But if you use Java Multithreading then you could execute robot's tasks in parallel if it is necessary.

```
Program:
        Task1       Task2
        Task3
```

If you notice, the human body executes several operations in parallel for example: Respiration, blood circulation, digestion, thinking and walking, besides the human brain process many data from our senses in parallel: sight, touch, smell, taste and hearing.

Most programming languages do not enable programmers to specify concurrent activities. Rather, the languages generally provide control statements that only enable programmers to perform one action at a time, proceeding to the next action after the previous one has finished. Historically, concurrency has been implemented with operating system primitives available only to experienced systems programmers.

2.2.- The Thread concept

A thread in Java is the minimum process's unit in parallelism terms. When you design a robot using Threads, you could think in the following ideas:

1. A thread to manage the locomotion subsystem (For a wheeled robot, biped robot, hexapod robot, etc…)
2. A thread to manage GPS
3. A thread to manage Bluetooth communications
4. A thread to manage the robot arm
5. A thread to manage the robot's senses (Ultrasonic sensors for example)

The robot's design process is iterative, because it is a complex task, but it is so useful becase when in the future you wan to incorporate a new feature then add a new task, a thread, or select the old thread and update this one with new capabilities.

In the following example, we will develop our first Thread which is managed by a main program.

HelloWorldThread.java
This is the first example using Threads. This thread prints in NXT LCD the message "Hello World" and the value of the variable i.

```
import lejos.nxt.*;

public class HelloWorldThread extends Thread{
	private int i = 0;

	public HelloWorldThread(){

	}

	public void run(){
		while(true){
			LCD.drawString("Hello World:", 0, 0);
			LCD.drawInt(i, 0, 1);
			LCD.refresh();
			i++;
		}
	}
}
```

ThreadTest1.java
This program executes the thread HelloWorldThread until you push the button ESCAPE in your NXT brick.

```
import lejos.nxt.*;

public class ThreadTest1 {
	private static HelloWorldThread hwt;

	public static void main(String[] args){
		hwt = new HelloWorldThread();
		hwt.start();

		while(!Button.ESCAPE.isPressed()){

		}

		System.exit(0);
	}
}
```

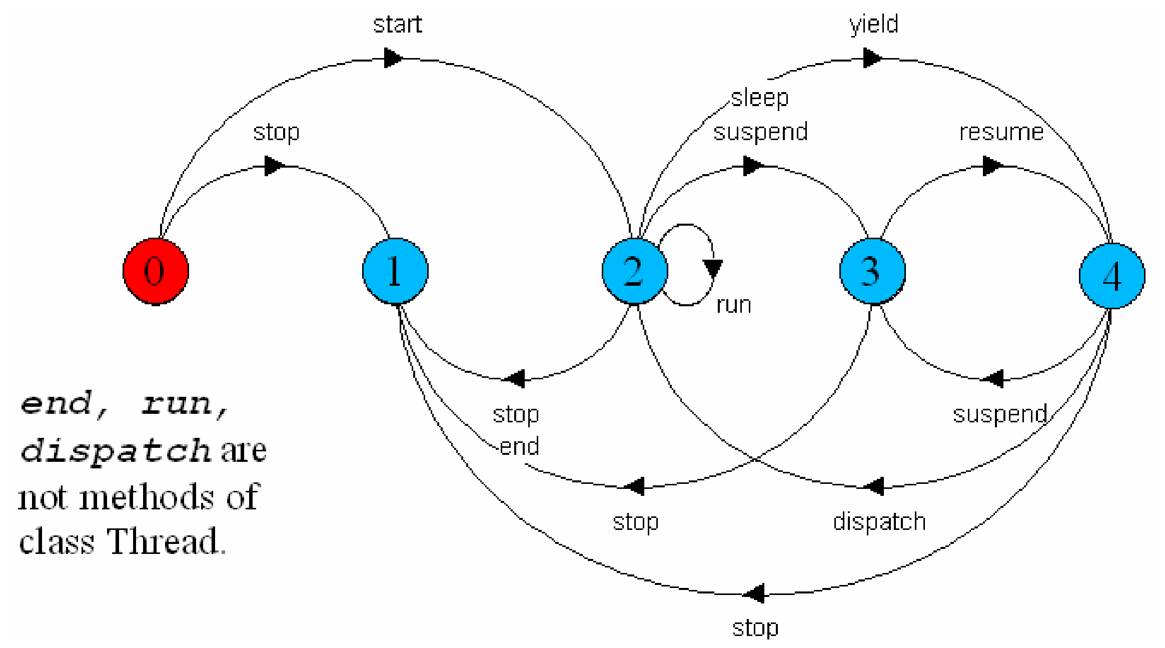
2.3.- The Thread life cycle

It is very important to know the life cycle of a Thread.

The possible states are the following:

- Start
- Sleep
- Interrupted

- Yield
- Join
- Interrupt



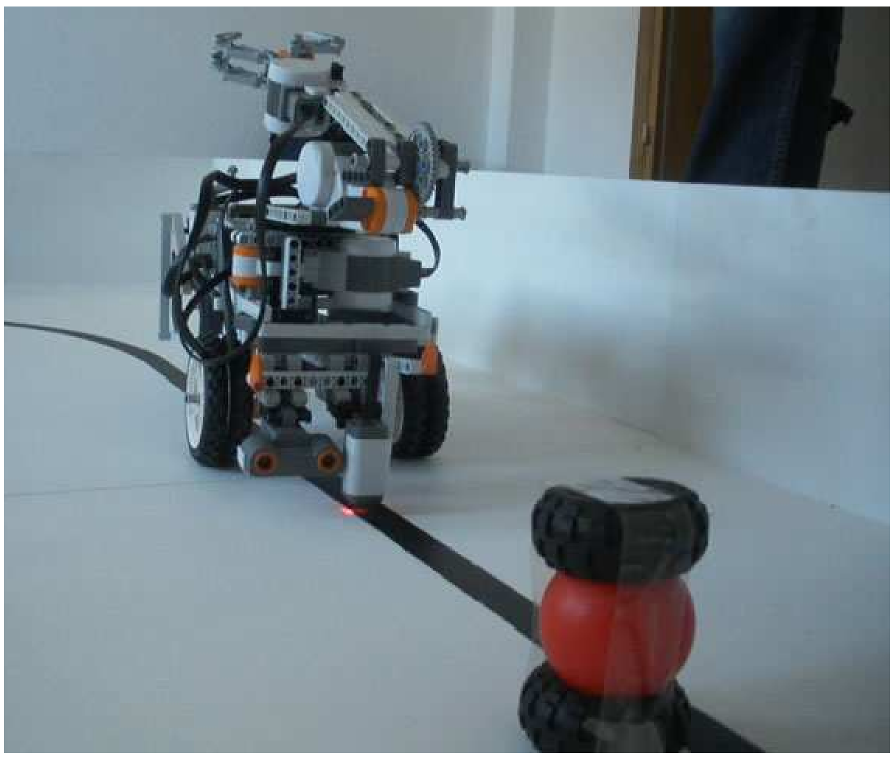
end, run, dispatch are not methods of class Thread.

2.4.- Examples

This section will explain Multithreading using real examples.

2.4.1.- Example1: LineFollower

Imagine if you have a NXT Robot which has the mission to follow a black line and if it discovers an obstacle then it stops and use its arm.



The original code which was coded without any Thread:

```
import lejos.nxt.*;

public class LineFollowerOld {

	public static void main(String[] args) {
		LightSensor ss = new LightSensor(SensorPort.S3);
		ss.setFloodlight(true);
		UltrasonicSensor us = new UltrasonicSensor(SensorPort.S1);

		while (us.getDistance()>25){
			LCD.drawInt(ss.readValue(), 3, 9, 0);
			LCD.refresh();

			while (ss.readValue()< 45){
				MotorPort.C.controlMotor(0, 3);
				MotorPort.B.controlMotor(80, 1);
			}

			while(ss.readValue()>=45){
				MotorPort.B.controlMotor(0, 3);
				MotorPort.C.controlMotor(80, 1);
			}
		}

		LCD.drawString("Object found!", 0,1);
		Sound.twoBeeps();
		Sound.twoBeeps();
	}
}
```

Once you analyze the tasks:

1. Follow black lines
2. Discover obstacles in the route

It is very easy to design and develop an scalable solutions with Threads:

The main Program:

```
import lejos.nxt.*;

public class HarisRobot {
	private static DataExchange DE;
	private static LineFollower LFObj;
	private static ObstacleDetector ODObj;

	public static void main(String[] args){
		DE = new DataExchange();
		ODObj = new ObstacleDetector(DE);
		LFObj = new LineFollower(DE);
		ODObj.start();
		LFObj.start();

		while(!Button.ESCAPE.isPressed()){
			//Empty
		}
		LCD.drawString("Finished",0, 7);
		LCD.refresh();
```

```
            System.exit(0);
        }
}
```

The thread used to follow a black line:

```
import lejos.nxt.*;

public class LineFollower extends Thread {
        DataExchange DEObj;

        private LightSensor ss;
        private UltrasonicSensor us;
        private final int colorPattern = 45;

        public LineFollower(DataExchange DE){
                DEObj = DE;

                ss = new LightSensor(SensorPort.S3);
                us = new UltrasonicSensor(SensorPort.S1);

                ss.setFloodlight(true);
        }

        public void run(){
                //Infinite Task
                while(true){
                        if(DEObj.getCMD() == 1){
                                if(ss.readValue()< colorPattern){
                                        MotorPort.C.controlMotor(0, 3);
                                        MotorPort.B.controlMotor(80, 1);
                                }else{
                                        MotorPort.B.controlMotor(0, 3);
                                        MotorPort.C.controlMotor(80, 1);
                                }
                                LCD.drawInt(ss.readValue(), 3, 9, 0);
                                LCD.refresh();
                        }else{
                                //Stop
                                MotorPort.B.controlMotor(0, 3);
                                MotorPort.C.controlMotor(0, 3);
                        }
                }
        }
}
```

The Thread used to detect Obstacles

```
import lejos.nxt.*;

public class ObstacleDetector extends Thread{
        private DataExchange DEObj;

        private UltrasonicSensor us;
        private final int securityDistance = 25;

        public ObstacleDetector(DataExchange DE){
                DEObj = DE;
                us = new UltrasonicSensor(SensorPort.S1);
        }
```

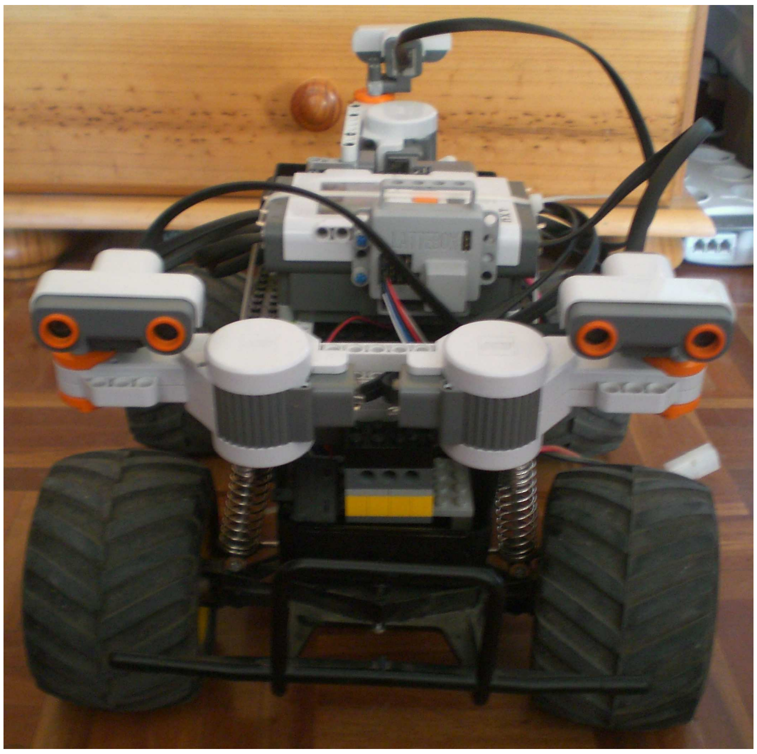
```
	public void run(){
		while(true){
			if(us.getDistance() > securityDistance){
				DEObj.setCMD(1);
			}else{
				DEObj.setCMD(0);

				//LCD Output
				LCD.drawString("Object found!", 0,1);
				LCD.refresh();
				Sound.twoBeeps();
				Sound.twoBeeps();
			}
		}
	}
}
```

Finally the Class used to exchange data among Threads:

```
public class DataExchange {
	//ObstacleDetector
	private boolean obstacleDetected = false;

	//Robot has the following commands: Follow Line, Stop
	private int CMD = 1;

	public DataExchange(){

	}

	/*
	 * Getters & Setters
	 */

	public void setObstacleDetected(boolean status){
		obstacleDetected = status;
	}

	public boolean getObstacleDetected(){
		return obstacleDetected;
	}

	public void setCMD(int command){
		CMD = command;
	}

	public int getCMD(){
		return CMD;
	}
}
```

2.4.2.- Example2: Stereo US

In the project RC Car managed by NXT Brick, http://www.juanantonio.info/jab_cms.php?id=228 exists 2 US Sensors which detect obstacles.



This subsystem runs in parallels with other subsystems as Navigation or GPS for example.

A tes program to test new subsystem for RC Car Project:

```
import lejos.nxt.*;

public class SUSETest {
	private static StereoUSEyes SUSEObj;

	public static void main(String[] args) throws Exception {
		TLBDataBridge TLBDB = new TLBDataBridge();
		SUSEObj = new StereoUSEyes(TLBDB);
		SUSEObj.start();

		while(!Button.ESCAPE.isPressed()){

		}
		LCD.drawString("Finished",0, 7);
		LCD.refresh();
		System.exit(0);
	}
}
```

Note: In a future I will code a Test Case for this subsystem using JUnit for leJOS

If notice StereoUSEye is based on 2 parts: Left Eye and Rigth Eye. The code to manage any eye is the following:

```
import lejos.nxt.*;

public class USEye extends Thread{
	private UltrasonicSensor US;
	private Motor motor;
	private boolean eyeSide = false;
	private boolean turnFlag = false;
	private int angles[] = {0,10,70,80};
	private int oldDistances[] = {0,0,0,0};
	private int distances[] = {0,0,0,0};

	static final boolean LEFTSIDE = true;
	static final boolean RIGHTSIDE = false;

	public USEye(UltrasonicSensor usObj,Motor mObj,boolean side){
		US = usObj;
		motor = mObj;
		eyeSide = side;
		motor.resetTachoCount();
		motor.setPower(100);
	}

	public void run(){
		//It is a infinite task
		while(true){
			SICKMode();
		}
	}

	public void SICKMode(){
		if(turnFlag){
			motor.rotateTo(0);
			oldDistances[0] = distances[0];
			distances[0] = US.getDistance();
			motor.rotateTo(10);
			oldDistances[1] = distances[1];
			distances[1] = US.getDistance();
			motor.rotateTo(70);
			oldDistances[2] = distances[2];
			distances[2] = US.getDistance();
			motor.rotateTo(80);
			oldDistances[3] = distances[3];
			distances[3] = US.getDistance();
			turnFlag = false;
		}else{
			motor.rotateTo(80);
			oldDistances[3] = distances[3];
			distances[3] = US.getDistance();
			motor.rotateTo(70);
			oldDistances[2] = distances[2];
			distances[2] = US.getDistance();
			motor.rotateTo(10);
			oldDistances[1] = distances[1];
			distances[1] = US.getDistance();
			motor.rotateTo(0);
			oldDistances[0] = distances[0];
			distances[0] = US.getDistance();
			turnFlag = true;
		}
	}

	public int[] getDistances(){
		return distances;
	}

	public int getDistance(int index){
		int distance = 0;
```

```
            if((index >= 0) && (index <= distances.length)){
                  distance = distances[index];
            }else{
                  distance = -1;
            }
            return distance;
      }
}
```

The class designed to manage 2 USEye object is the following:

```
import lejos.nxt.*;

public class StereoUSEyes extends Thread{
      private TLBDataBridge TLBDB;

      private USEye leftEye;
      private USEye rightEye;
      private int[] leftDistances;
      private int[] rightDistances;
      private final int[] angles = {0,10,70,80};

      public StereoUSEyes(TLBDataBridge _TLBDB){
            TLBDB = _TLBDB;

            UltrasonicSensor leftUS = new
UltrasonicSensor(SensorPort.S1);
            UltrasonicSensor rightUS = new
UltrasonicSensor(SensorPort.S2);
            Motor leftMotor = Motor.A;
            Motor rigthMotor = Motor.B;
            leftEye = new USEye(leftUS,leftMotor,USEye.LEFTSIDE);
            rightEye = new USEye(rightUS,rigthMotor,USEye.RIGHTSIDE);

      }

      public void run(){
            leftEye.start();
            rightEye.start();

            //Set Enabled the subsystem
            TLBDB.setEyesEnabled(true);

            while(true){

                  leftDistances = leftEye.getDistances();
                  rightDistances = rightEye.getDistances();
                  TLBDB.setLeftEyedistances(leftDistances);
                  TLBDB.setRightEyedistances(rightDistances);

                  /*
                  for(int i=0; i< leftDistances.length;i++){
                        LCD.drawInt(angles[i], 0, i);
                        LCD.drawString("   ", 4, i);
                        LCD.drawInt(leftDistances[i], 4, i);
                        LCD.drawString("   ", 10, i);
                        LCD.drawInt(rightDistances[i], 10, i);
                  }
                  LCD.refresh();
                  //Exchange data with other subsystems
                   *
```

```
                        */
                }
        }
}
```

If you want to observe SUSE in action, see the following video on Youtube:
http://www.youtube.com/watch?v=3hXTeK8lFMc

3.- Links

3.1.- Multithreading Links

http://sol.cs.fhm.edu/rs/rcx-threads/index.html
http://www.cs.fit.edu/~ryan/java/language/thread.html
http://profesores.elo.utfsm.cl/~agv/elo329/Java/threads/JavaThreads.html
http://www.cs.wustl.edu/~schmidt/patterns-ace.html
http://gee.cs.oswego.edu/dl/cpj/