Threads in Java and Lejos

Henning Christiansen

Robotics course

March 7, 2018

Threads, what and why

What: (simulated) parallel processes

Why: More natural way of programming (e.g., for robots)

- keep track of, control and react to many thing at the same time
- “... as living creatures do ...”

Underlying system (scheduler) takes care that all processes (threads) gets a fair share of available CPU time (hopefully!).

Primary applications of threads for robots

- high-level sensors (today's exercise)
- behaviours running in parallel
    - implemented using threads; much easier to understand and to use when you know threads
    - later in the course

Threads in Java

```
public class MyNewThread extends Thread {
    public MyNewThread() {
        //  initialize internal and perhaps shared data structures
    }
    public void run() {
        //  supplied by programmer; used by system ("scheduler")
        //  series of actions; typically a loop
    }
}


...
MyNewThread t = new MyNewThread();
...
r.start();
```

Timing and sharing time

Threads must terminate for program to terminate

- run terminates by itself; exception or error; "depricated **`stop()`**"

Or use (yet another Java misnomer) **`daemon`** threads

Good practice for **`run()`**

```
while (true) {     // or something true for a while
        // delay for a short moment
        // sense
        // act
}
```

Perhaps also a **`yield();`**

Example: A beeper thread (1:2)

File: Thread1.java

```
...
 public static class IRObjectCounterThread extends Thread {
   //the following copy-pasted from the TestIR program
   EV3IRSensor infraRed;
   SampleProvider infraRedDistanceProvider;

   public IRObjectCounterThread(){
      infraRed = new EV3IRSensor(SensorPort.S2);
      infraRedDistanceProvider = infraRed.getMode("Distance");
      this.setDaemon(true); // Without this, the program won't stop
      }

   public void run() {
      while(true){
         Sound.pause(50); // needs no try-catch as do sleep
         float [] sample = new float[infraRedDistanceProvider.sampleSize()] ;
         infraRedDistanceProvider.fetchSample(sample,0);
         int d = (int) sample[0];
         if(d<100) Sound.playTone(2000-d*20,200);
} } }
```

Example: A beeper thread (2:2)

File: Thread1.java

```
public class Thread1{
  public static void main(String[] args){
     IRObjectCounterThread th =
          new IRObjectCounterThread();
     th.start();

     while(!Button.ESCAPE.isDown()){
        try { Thread.sleep(100);}
        catch (InterruptedException e) {
             e.printStackTrace();}
       }
    }
```

Example: As before but cleaned up a little

File: Thread2.java

```
public class Thread2{
  public static void main(String[] args){
     IRObjectCounterThread th =
            new IRObjectCounterThread();
     th.start();

     while(!Button.ESCAPE.isDown()){
       try { Thread.sleep(100);}
       catch (InterruptedException e) {
            e.printStackTrace();}
 } }
```

Example: As before but cleaned up a little

File: Thread2.java

```
public class Thread2{
  public static void main(String[] args){
     IRObjectCounterThread th =
            new IRObjectCounterThread();
     th.start();

     while(!Button.ESCAPE.isDown()){
        try { Thread.sleep(100);}
        catch (InterruptedException e) {
             e.printStackTrace();}
 } }
```

```
public static void startDaemons() {
     IRObjectCounterThread th =
         new IRObjectCounterThread();
     th.start();
    }
```

Example: As before but cleaned up a little

File: Thread2.java

```
public class Thread2{
  public static void main(String[] args){
    startDaemons();


     while(!Button.ESCAPE.isDown()){
       try { Thread.sleep(100);}
       catch (InterruptedException e) {
            e.printStackTrace();}
 } }
```

```
public static void startDaemons() {
     IRObjectCounterThread th =
          new IRObjectCounterThread();
     th.start();
    }
```

Example: Adding another thread: More beeping (1:3)

File: Thread3.java

```
public class Thread3{
  public static void main(String[] args){
    startDaemons();


     while(!Button.ESCAPE.isDown()){
       try { Thread.sleep(100);}
       catch (InterruptedException e) {
            e.printStackTrace();}
 } }
```

Example: Adding another thread: More beeping (1:3)

File: Thread3.java

Main method exactly the same :)

```
public class Thread3{
  public static void main(String[] args){
    startDaemons();

     while(!Button.ESCAPE.isDown()){
       try { Thread.sleep(100);}
       catch (InterruptedException e) {
            e.printStackTrace();}
 } }
```

Example: Adding another thread: More beeping (2:3)

File: Thread3.java

No surprise at all

```
public static class TouchBeeperThread extends Thread {
    EV3TouchSensor touch;
    SampleProvider touchProvider;

    public TouchBeeperThread() {
        touch = new EV3TouchSensor(SensorPort.S1);
        touchProvider = touch.getTouchMode();
         this.setDaemon(true);
    }
    public void run() {
      while(true){
        Sound.pause(50);
        float [] sample = new float[touchProvider.sampleSize()] ;
        touchProvider.fetchSample(sample,0);
        int yesNo = (int) sample[0];
         if(yesNo>0.5) Sound.playTone(2000,400);
} } }
```

Example: Adding another thread: More beeping (3:3)

File: Thread3.java

Only one thing left to make it work

```
public static void startDaemons() {
    IRObjectBeeperThread th1 =
          new IRObjectBeeperThread();
    TouchBeeperThread th2 =
          new TouchBeeperThread();
    th1.start();
    th2.start();
   }
```

Threads and common variables

(a bit more than we need for our robots)

- Requires a bit of care to get things right
- A classical database example: two transactions updating the same account:

Add my salary

```
temp1 = balance;

newBalance1 = balance+10^5;

balance = newBalance1;
```

Draw a payment

```
temp2 = balance;

newBalance1 = balance - 50;

balance = newBalance1;
```

Threads

(a bit r

- Requires a
- A classical
  updating th

Add my sala

```
temp1 = balanc

newBalance1 = 

balance = newB
```

But

```
temp1 = balance;

                temp2 = balance;

newBalance1 = balance+10^5;

balance = newBalance1;

                newBalance1 = balance - 50;

                balance = newBalance1;
```

Java offers tools for that

- *Atomicity* (as it is called in the DB business) provided by:

```
synchronized (<some object>) {
    statement-1;
    statement-2;
    ...
}
```

Typically the "some object" is the common data structure, but not always...

See more details in the Java tutorial

Example: Working on a common variable (1:3)

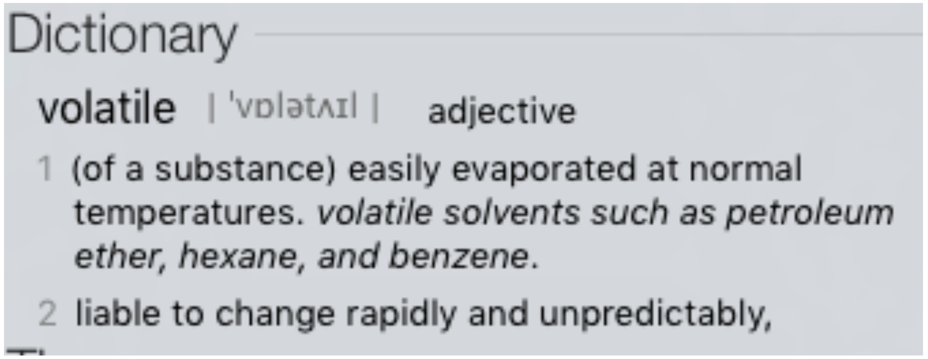
File: Thread4.java

Java when most grotesque – but it works!

Declaring the variable suitable with a "lock object"

```
public static volatile int count=0;

public static Object countLockObject =
             new Object();
```



Dictionary

volatile | ˈvɒlətʌɪl | adjective

1 (of a substance) easily evaporated at normal temperatures. *volatile solvents such as petroleum ether, hexane, and benzene.*

2 liable to change rapidly and unpredictably,

In Javanesian:

- no caching; mapped to main memory at once

Example: Working on a common variable (2:3)

File: Thread4.java

Add code for counting beeps in each of the beeper threads E.g.

```
public static class IRObjectBeeperThread extends Thread {
....
    public void run() {
      if(...) {Sound.playTone(......);
                   // synchronization is probably not needed,
                    // but who will trust "probably"
                   synchronized(countLockObject){count++;}}
```

Example: Working on a common variable (3:3)

File: Thread4.java

Adding a third thread for printing out no. of beeps from time to time.

```
public static class PrintNumberOfBeepsNowAndThen extends Thread
{
   public PrintNumberOfBeepsNowAndThen() {
      this.setDaemon(true);}

    public void run() {
       while(true){
         try { Thread.sleep(5000);}
         catch (InterruptedException e) {e.printStackTrace();}
         System.out.println("Number of beeps: "+count);

}}}
```

Topic for our next exercise/ assignment: High-level sensors

- a software handle that makes it possible for a program to check properties that usually requires cognitive skills
- depends on low-level sensing and/or internal state
- requires some code lines for interpretating the low level measurements and "translate" them into high-level information
- In LeJOS: we ***might*** adapt the style use for LeJOS' normal sensors, but ***a better approach*** is to make our own, more intuitive and programmer friendly style

The dubious LeJOS - Java style

```
EV3TouchSensor touch =
    new EV3TouchSensor(SensorPort.S1);
SampleProvider touched = touch.getTouchMode();
float[] sample = new
           float[touched.sampleSize()];
....
touched.fetchSample(sample,0);
```

The dubious LeJOS - Java style

```
My Sensor mySensor =
    new My Sensor( () );
SampleProvider sensed = mySensor.g myMode Mode();
float[] sample = new
            float[touched.sampleSize()];

....

sensed.fetchSample(sample,0);
```

The style advocated by your teacher

- The exercise: Make a bumpy-road sensor; road is bumpy if more than 5 bumps in the last two sec's

```
public class BumpyRoadSensor extends Thread {

...

  public BumpyRoadSensor() {...}

  public void run() {while(true){...}}

  public boolean roadIsBumpy() {...}

}
```