Fortsetzung
Java unter leJOS

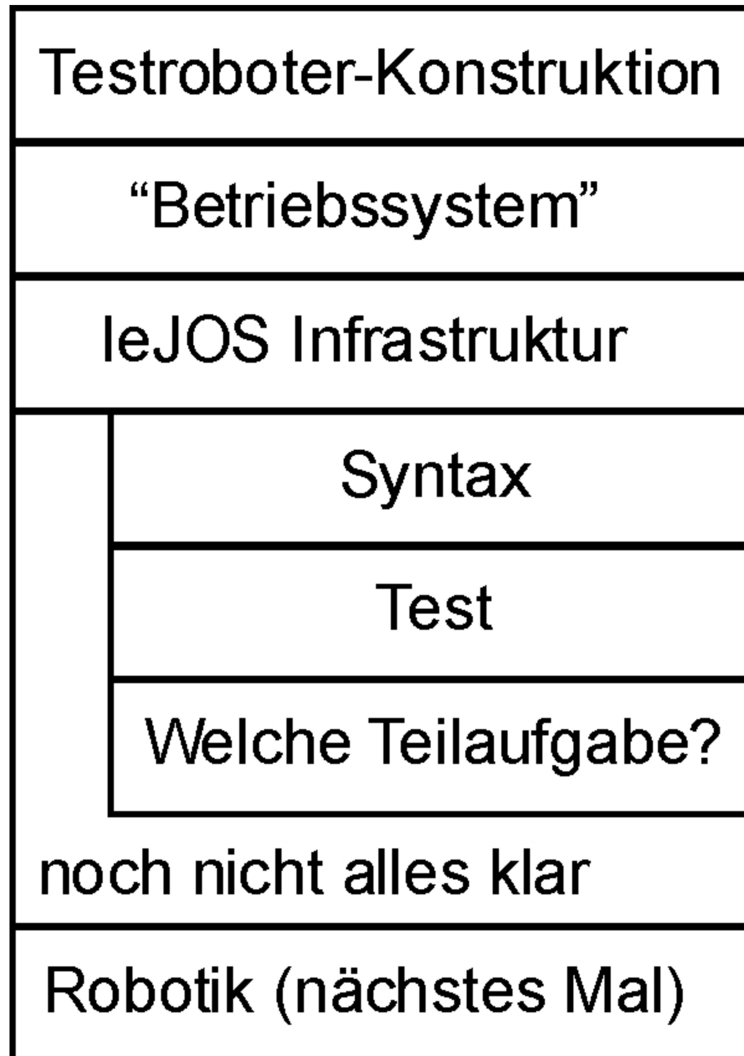
für den LEGO®-Roboter-Wettbewerb

Oktober/November 2002

Prof. Dr. Michael Wülker
Fachhochschule Offenburg

Überblick

FachHochschule Offenburg University of Applied Sciences



erste Schritte

- HelloRobby
- Variable (Attribute) und Operatoren
- Klassen, Objekte und Vererbung
- Kontrollstrukturen
- Motoren und Sensoren

- Klassenstruktur von leJOS
- Schnittstellen
- Parallelrechner-Betrieb
  Listener, Behavior, Threads
- Nützliches: Timer, Sound, Display
- Kommunikation: Datalogging

Fortsetzung

Pakete:
funktionale Aufteilung im Großen

FachHochschule Offenburg University of Applied Sciences

Standardpakete von Java

Interfaces: Schnittstellen

[All Classes](#)

Packages

java.io
java.lang
java.net
java.util
javax.servlet.http
josx.platform.rcx
josx.rcxcomm
josx.robotics
josx.util

josx.platform.rcx

Interfaces

ButtonListener
LCDConstants
ListenerCaller
Opcode
Segment
SensorConstants
SensorListener
SerialListener

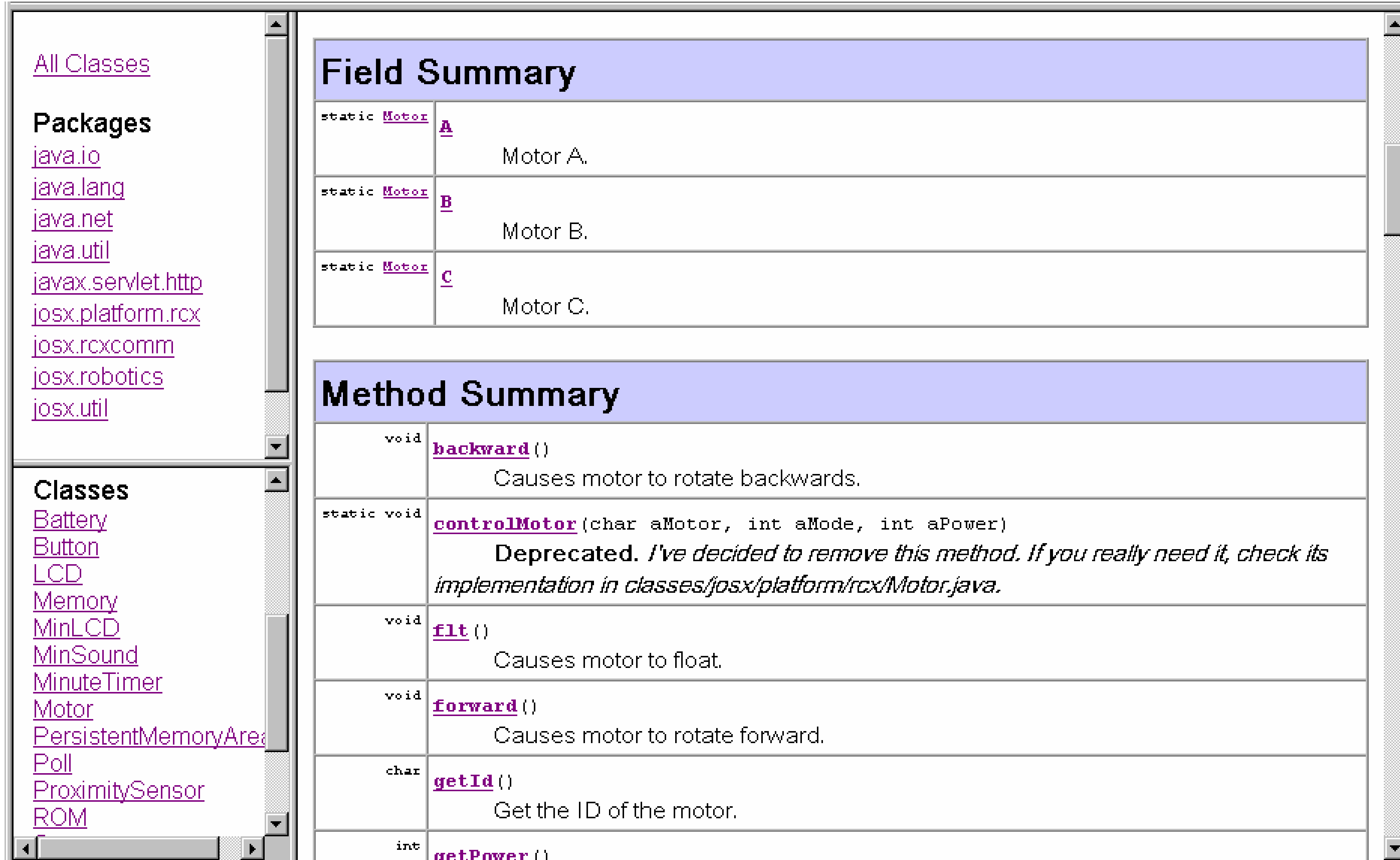
Classes

Class Summary	
Battery	Provides access to Battery.
Button	Abstraction for an RCX button.
LCD	LCD routines.
Memory	Provides access to memory.
MinLCD	Only the most basic APIs from LCD.
MinSound	Only the most basic APIs from Sound.
MinuteTimer	Provides access to Battery.
Motor	Abstraction for a motor.
PersistentMemoryArea	A memory area for persistent storage.
Poll	Provides blocking access to events from the RCX.
ProximitySensor	A 'sensor' to detect object proximity.
ROM	Provides access to ROM routines.
Sensor	Abstraction for a sensor (*considerably changed since alpha5*).
Serial	Low-level API for infra-red (IR) communication between an RCX and the IR tower or between two RCXs.
Servo	Implmentation of a servo using a Motor and a Rotation Sensor.
Sound	RCX sound routines.
TextLCD	Display text on the LCD screen.

Methoden innerhalb einer Klasse funktionale Aufteilung, Modularität

Attribute und Methoden

"Unter-programme": innere Methoden



Schnittstellen

Eine Schnittstelle entspricht einer Klasse, die keine Attribute und ausschließlich abstrakte, d. h. nicht implementierte, Operationen besitzt.

Beispiel: Schnittstelle **`TimerListener`** in **`josx.util`**

Method Summary	
void	timedOut() Called every time the Timer fires.

Quelltext

```
public interface TimerListener {
  public void timedOut();
}
```

- Schnittstellen ermöglichen eine **spezifizierende Sicht**, da sie nur die Signaturen (Methodenköpfe) festlegen und keine Aussagen über die Implementierung gemacht werden.
- Bei der Implementierung ist man gezwungen, **alle** Methoden zu implementieren.
- Eine Schnittstelle ist ein **Referenztyp**. Von ihm können Referenzvariable gebildet werden, die auf Objekte zeigen, deren Klassen die Schnittstelle implementieren.
- Werden Schnittstellentypen als Übergabeparameter an Operationen verwendet, kann so eine **Typüberprüfung** erzwungen werden.

Timer als Beispiel, wie man eine Schnittstelle benutzt

Constructor Summary
Timer(int theDelay, TimerListener el) Create a Timer object.

Quelltext der Schnittstelle:

```
public interface TimerListener {
  public void timedOut();
}
```

Implementierung der Schnittstelle in **SoundingTimerListener.java**

```
import josx.platform.rcx.*;
import josx.util.*;
public class SoundingTimerListener
            implements TimerListener {
  public void timedOut() {
    Sound.beep();
  }
}
```

TimerTest.java

```
import josx.platform.rcx.*;
import josx.util.*;
class TimerTest {
  private static SoundingTimerListener chime;
  private static Timer timePiece;
  public static void main(String args[]) {
    chime = new SoundingTimerListener();
    int beat = 1000;    // start with 1 s beats
    timePiece = new Timer(beat, chime);
    for (int i = 0; i<6; i++) {
      timePiece.setDelay(beat);
      LCD.showNumber(beat);
      timePiece.start();
      pause(beat*10);
      timePiece.stop();
      pause(1000);
      beat /= 2;
    }
    System.exit(1);
  }
  private static void pause(int delay) {
    ....
  }
}
```

Nebenläufigkeit – Listener laufen in separatem Thread

FachHochschule Offenburg University of Applied Sciences

```
import josx.platform.rcx.*;
public class SenseBot extends TurnBot {
  private Sensor leftTouch = Sensor.S1;
  private Sensor rightTouch = Sensor.S3;
  public SenseBot() {
    leftTouch.addSensorListener(new TouchBackListener(this));
    rightTouch.addSensorListener(new TouchBackListener(this));
  }
}
```

Klasse trägt sich als Listener für eine bestimmten Sensor mit **`addSensorListener`** ein.

`TouchBackListener` implementiert **`SensorListener`**.

```
import josx.platform.rcx.*;
public class TouchBackListener
            implements SensorListener {
  private TurnBot robby;
  public TouchBackListener() { }
  public TouchBackListener(TurnBot robby) {
    this.robby = robby;
  }
  public void stateChanged(Sensor s,
                int oldValue, int newValue) {
    if (s.readBooleanValue()) {
      robby.reverseDirection();
      Sound.systemSound(false, 3);
      robby.pause(500);
      robby.stop();
      TextLCD.print("wall!");
    }
  }
}
```

Obwohl das Programm pausiert, wird auf einen Tastendruck reagiert! **`SensorListener`** läuft in einem eigenen "Thread":

```
import josx.platform.rcx.*;
class SenseBotTest {
  static SenseBot touchyRobby;
  public static void main(String args[]) {
    touchyRobby = new SenseBot();
    touchyRobby.go_forward();
    touchyRobby.pause(20000);
  }
}
```

Linienverfolgung mit Hilfe eines SensorListeners

FachHochschule Offenburg University of Applied Sciences

```
import josx.platform.rcx.*;
class LineBotTest {
  static LineBot lineRobby;
  public static void main(String args[]) {
    lineRobby = new LineBot();
    lineRobby.go();
  }
}
```

```
import josx.platform.rcx.*;
public class LineBot extends SenseBot {
  Sensor lightSensor = Sensor.S2;
  public LineBot() {
    super();
    lightSensor.activate();
    lightSensor.addSensorListener(
      new LineFollowingListener(this));
  }
  public void go() {
    TextLCD.print("LINE");
    this.go_forward();
    while (true) {
    }
  }
  public void correct_course() {
    go_backward(); pause(150);
    veer_left(); pause(80);
    go_forward();
  }
}
```

Aufruf des Konstruktors der Oberklasse; aktiviert den **TouchBackListener**!

SensorListener für den Lichtsensor

Änderung in **TurnBot.java**

```
public void veer_left() {
  leftWheel.backward();
  rightWheel.forward();
}
```

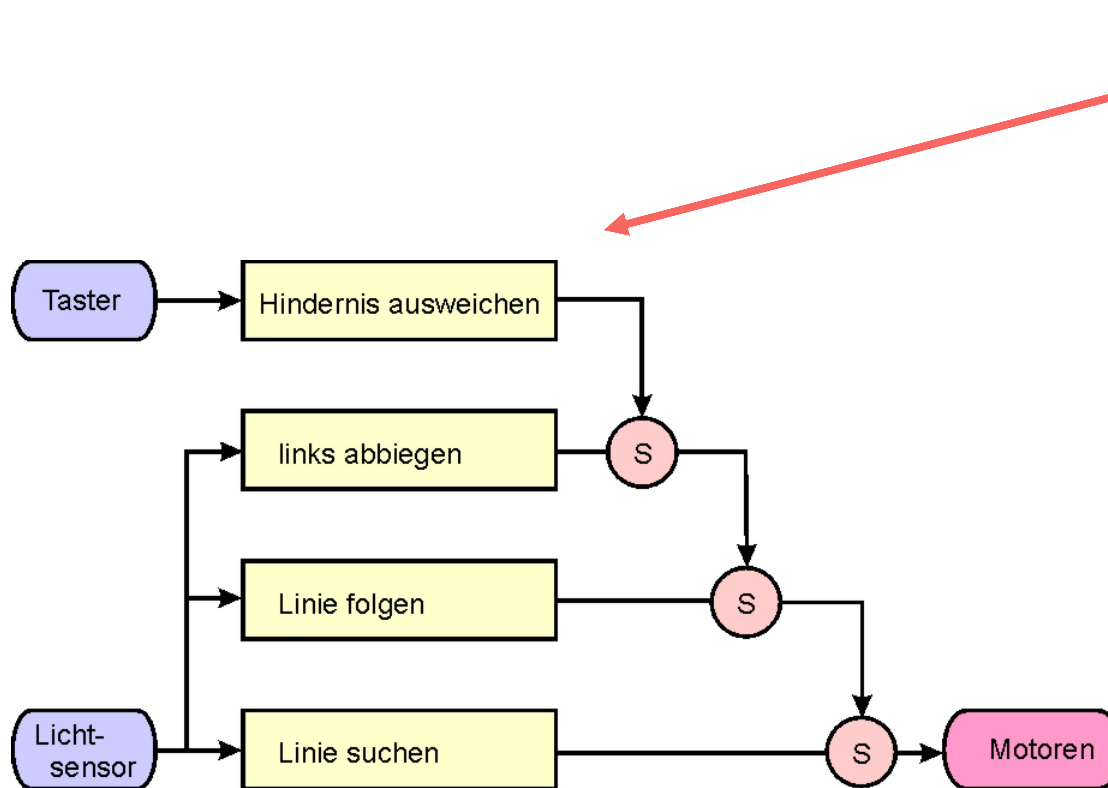
Nebenläufigkeit

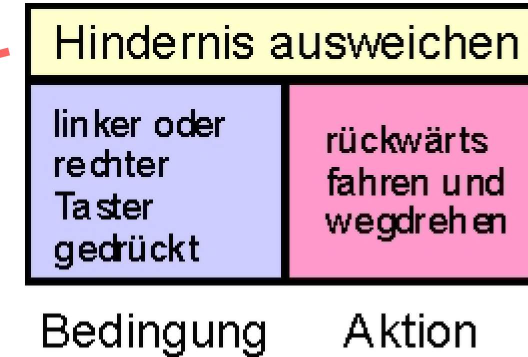
```
import josx.platform.rcx.Sensor;.
import josx.platform.rcx.SensorListener;.
public class LineFollowingListener.
              implements SensorListener {.
  private LineBot lineRobby;.
  private int threshold;.
  public LineFollowingListener() { }.
  public LineFollowingListener(LineBot robby) {
    lineRobby = robby;.
    threshold = 0;.
    lineRobby.pause(300);.
  }.
  public void stateChanged(Sensor s,.
               int oldValue, int newValue) {.
    if (threshold == 0).
      threshold = newValue + 10;.
    if (newValue > threshold).
      lineRobby.correct_course();.
  }.
}.
```

Für die notwendige Kurskorrektur wird eine Methode von **`LineBot`** benutzt.

R. Brooks: Behavior Control (Subsumption Architecture)

FachHochschule Offenburg University of Applied Sciences






```
public class BehaviorTest {
  public static void main(String args[]) {
    TurnBot robby = new TurnBot();
    Behavior b1 = new SearchLine((Robot)robby);
    Behavior b2 = new FollowLine(robby);
    Behavior b3 = new BranchOff(robby);
    Behavior b4 = new TouchBack(robby);
    // priority increases with the index.
    Behavior [] barray = {b1, b2, b3, b4};
    Arbitrator arby = new Arbitrator(barray);
    arby.start();
  }
}
```

Hierarchie von Verhaltensmustern:
Überlebenswichtige Funktion unterdrückt weniger wichtige Aktivitäten

Behavior: Linie suchen

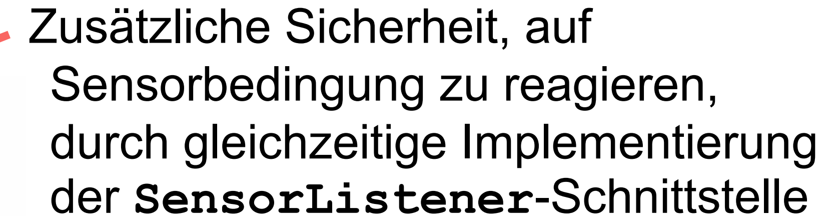

```
import josx.platform.rcx.*;
import josx.robotics.*;
public class SearchLine implements Behavior, SensorConstants {
  private Robot robby;
  public SearchLine(Robot robi) {
    robby = robi;
    Sensor.S2.setTypeAndMode(SENSOR_TYPE_LIGHT, SENSOR_MODE_PCT);
    Sensor.S2.activate();
  }
  public boolean takeControl() {
    if (Sensor.S2.readValue() > GeneralConstants.whiteThreshold) {
      LCD.showNumber(1111); return true;
    }
    return false;
  }
  public void suppress() {
    robby.stop();
  }
  public void action() {
    Motor.A.forward(); Motor.C.forward();
  }
}
```

Bedingung

Unterdrückungs-
mechanismums

Aktionsablauf

Behavior: links abbiegen



Zusätzliche Sicherheit, auf Sensorbedingung zu reagieren, durch gleichzeitige Implementierung der **`SensorListener`**-Schnittstelle

```
import josx.platform.rcx.*;
import josx.robotics.*;
public class BranchOff
      implements Behavior, SensorListener,
                 SensorConstants, GeneralConstants {
  private TurnBot robby;
  private boolean hitBlack;
  public BranchOff(TurnBot robi) {
    Sensor.S2.setTypeAndMode(SENSOR_TYPE_LIGHT, SENSOR_MODE_PCT);
    Sensor.S2.addSensorListener(this);
    hitBlack = false; robby = robi;
  }
  public void stateChanged(Sensor bumper,
    int oldValue, int newValue) {
    if (newValue < blueThreshold)
      hitBlack = true;
  }
  public boolean takeControl() { ... }
  public void suppress() { ... }
  public void action() { ... }
}
```

Behavior-Methoden für **`BranchOff`**

FachHochschule Offenburg University of Applied Sciences

```
public boolean takeControl() {
  if (Sensor.S2.readValue() < blueThreshold || hitBlack) {
    hitBlack = false; LCD.showNumber(3333);
    return true;
  }
  else
    return false;
}
public void suppress() {
  robby.stop();
}
public void action() {
  robby.veer_left(); robby.pause(500);
  robby.veer_left(); robby.pause(500);
  robby.go_forward();
}
```

Ergebnis des **`SensorListeners`** wird mit berücksichtigt.

Impliziter Thread: Motoren bewegen sich unabhängig vom Programmablauf weiter.

Nebenläufigkeit: explizite Threads


```
public class FollowLine implements Behavior {
  public static boolean act;
  private TurnBot robby;
  private Sensor lightSensor;
  public FollowLine(TurnBot robi) {
    robby = robi; lightSensor = Sensor.S2;
    lightSensor.setTypeAndMode(
      SensorConstants.SENSOR_TYPE_LIGHT,
      SensorConstants.SENSOR_MODE_PCT);
    lightSensor.activate(); act = false;
    (new LineFollowerThread(robby)).start();
  }
  public boolean takeControl() {
    if (lightSensor.readValue() <
            GeneralConstants.whiteThreshold) {
      LCD.showNumber(2222); return true;
    }
    else
      return false;
  }
  public void suppress() {
    act = false; robby.stop(); }
  public void action() {
    robby.go_forward(); act = true; }
}
```

Die Aktionsfolge, um der Linie zu folgen, muss unabhängig vom Hauptthread, in dem der **`Arbitrator`** die **`takeControl()`**-Methoden zyklisch abfragt, ablaufen. Sonst blockiert das Verfolgen der Linie den **`Arbitrator`**!

Thread wird sofort gestartet.

Durch das Flag **`act`** wir dem **`LineFollowerThread`** signalisiert, ob er agieren oder schlummern soll.

Nebenläufigkeit: explizite Threads


```
public class LineFollowerThread extends Thread {
  private TurnBot robby;
  public LineFollowerThread(TurnBot robbi) {
    robby = robbi;
  }
  public void run() {
    while (true) {
      if (FollowLine.act) {
        SearchLine.setNewGo();
        if (Sensor.S2.readValue() > GeneralConstants.whiteThreshold) {
          robby.go_backward();
          try {this.sleep(80);} catch (InterruptedException ie) {}
          robby.veer_left();
          try {this.sleep(120);} catch (InterruptedException ie) {}
          robby.go_forward();
        }
      }
      else {
        try {this.sleep(100);} catch (InterruptedException ie) {}
      }
    }
  }
}
```

Ein expliziter Thread muss die **`run()`**-Methode implementieren.

`act` wird in einer Endlosschleife abgeprüft.

Nützliches: Sound zum Debuggen

FachHochschule Offenburg University of Applied Sciences

```
import josx.platform.rcx.*;
class playSoundTest {
  public static void main(String args[]) {
    int msToWait = 1000;
    Sound.beep(); pause(msToWait);
    Sound.twoBeeps(); pause(msToWait);
    Sound.beepSequence(); pause(msToWait);
    Sound.buzz(); pause(msToWait);
    for (int i=0; i<6; i++) {
      Sound.systemSound(true, i);
      pause(msToWait);
    }
  }
  private static void pause(int delay) {
    ....
  }
}
```

Abspielen von allen Systemtönen über die **`systemSound()`**-Methode möglich.

`playTone()`-Methode legt die Tonhöhe in Hz und die Dauer in cs (Centisekunden) fest. **`pause()`** wartet, bis der Ton abgespielt ist, da der Soundgenerator unabhängig vom Prozessor arbeitet.

```
import josx.platform.rcx.*;
class playToneTest {
  public static void main(String args[]) {
    int msToWait = 1000;
    Sound.playTone(262, 40); pause(500);
    Sound.playTone(294, 40); pause(500);
    Sound.playTone(330, 40); pause(500);
    Sound.playTone(294, 40); pause(500);
    Sound.playTone(262, 160); pause(2000);
    pause(2000);
  }
  private static void pause(int delay) {
    ....
  }
}
```

Sound vom "Klavier" in BricxCC

FachHochschule Offenburg University of Applied Sciences

Wegen der Ähnlichkeiten in der Syntax von Not-Quite-C (NQC) und Java können die Daten vom "Klavier" des Bricx-Command-Centre übernommen werden.

In NQC arbeitet **`Wait()`** in Centisekunden.

```
import josx.platform.rcx.*;
class playPianoMusic {
  private static final int __NOTETIME = 10;
  private static final int __WAITTIME = 12;
  public static void main(String args[]) {
    Sound.playTone(523,4*__NOTETIME); Wait(4*__WAITTIME);
    Sound.playTone(659,2*__NOTETIME); Wait(2*__WAITTIME);
    Sound.playTone(523,2*__NOTETIME); Wait(2*__WAITTIME);
    Sound.playTone(392,4*__NOTETIME); Wait(4*__WAITTIME);
    Sound.playTone(392,4*__NOTETIME); Wait(4*__WAITTIME);
    Sound.playTone(392,2*__NOTETIME); Wait(2*__WAITTIME);
    Sound.playTone(523,2*__NOTETIME); Wait(2*__WAITTIME);
    Sound.playTone(392,2*__NOTETIME); Wait(2*__WAITTIME);
    Sound.playTone(330,2*__NOTETIME); Wait(2*__WAITTIME);
    Sound.playTone(262,4*__NOTETIME); Wait(4*__WAITTIME);
    Wait(100);
  }
  private static void Wait(int delay) {
    try {
      Thread.sleep(delay*10);
    }
    catch (InterruptedException ie) {
      TextLCD.print("error");
    }
  }
}
```

Nützliches: LCD-Display zum Debuggen

FachHochschule Offenburg University of Applied Sciences

leJOS erlaubt es die LCD-Anzeige segmentweise anzusprechen.

Textanzeige in RCXDirectMode der RCXTools zeigt, wie die Buchstaben des Alphabets mit den beschränkten Mitteln einer 7-Segment-Anzeige dargestellt werden können.

```
import josx.platform.rcx.*;
class DisplayTest {
  public static void main(String args[]) {
    Sensor.S1.setTypeAndMode(SensorConstants.SENSOR_TYPE_TOUCH,
                             SensorConstants.SENSOR_MODE_BOOL);
    LCD.clear(); LCD.setSegment(Segment.SENSOR_1_VIEW);
    if(Sensor.S1.readBooleanValue())
      LCD.setSegment(Segment.SENSOR_1_ACTIVE);
    LCD.refresh();
    LCD.showNumber(Sensor.S1.readRawValue()); pause(1000);

    LCD.clear(); LCD.setSegment(Segment.MOTOR_A_VIEW);
    if (Motor.A.isMoving()) {
      TextLCD.print("A on");
      if (Motor.A.isForward())
        LCD.setSegment(Segment.MOTOR_A_FWD);
      else if (Motor.A.isBackward())
        LCD.setSegment(Segment.MOTOR_A_REV);
    }
    else
      TextLCD.print("A off");
    LCD.refresh(); pause(1000);
    System.exit(1);
  }
  private static void pause(int delay) { ... }
}
```

Datenlogging

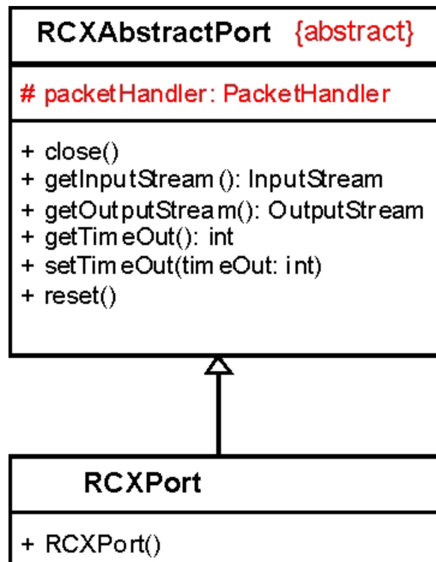

```
import java.io.*;
import josx.rcxcomm.*;
import josx.platform.rcx.*;
public class Datalog {
  private int nSamples;
  private int indexSample = 0;
  private short[] data;
  public Datalog() {
    data = new short[50];
    nSamples = 50;
  }
  public Datalog(int nSamples) {
    data = new short[nSamples];
    this.nSamples = nSamples;
  }
  public void AddToDatalog(short sample) {
    data[indexSample] = sample;
    indexSample++;
  }
  public int getDatalogLength() {
    return indexSample;
  }
```

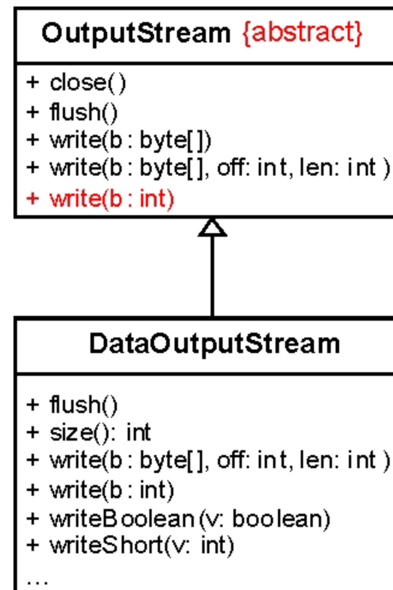
Voreinstellung der Datalog-Länge

Überladung des Standardkonstruktors zur gezielten Einstellung der Datalog-Länge

Datenlogging

Ausgabe der Daten erfolgt über den **RCXPort**, über den ein **DataOutputStream** gesendet wird.






```
public void uploadDatalog() {
  uploadDatalog(getDatalogLength());
}
public void uploadDatalog(int length) {
  TextLCD.print("upld");
  RCXPort port = null;
  try {
    port = new RCXPort();
    DataOutputStream out =
      new DataOutputStream(port.getOutputStream());
    out.writeShort(length);
    for (int k=0; k<length; k++)
      out.writeShort(data[k]);
    out.flush();
  } catch (IOException ioE) {
    TextLCD.print("uperr");
  } finally {
    port.close();
  }
  LCD.clear();
}
}
```

Empfang des Datalogs auf dem PC

FachHochschule Offenburg University of Applied Sciences

```
import josx.rcxcomm.*;
import java.io.*;
import java.text.*;
import java.util.*;
class DatalogReceiverTestOnPC {
  public static void main(String args[]) {
    try {
      RCXPort port = new RCXPort();
      InputStream is = port.getInputStream();
      DataInputStream dis = new DataInputStream(is);
      int length = dis.readShort();
      int[] dataLogged = new int[length];
      for(int i=0;i<length;i++) {
        int n = dis.readShort();
        dataLogged[i] = n;
      }
```

Beim Empfang der Daten auf dem PC steht wiederum **`RCXPort`** zur Verfügung und es wird ein **`DataInputStream`** verwendetet, der Unterklasse von **`InputStream`** ist.

Schreiben der aufgenommenen Daten in eine Datei

FachHochschule Offenburg University of Applied Sciences

Die Daten werden in eine Textdatei geschrieben und können dann mit entsprechenden Programmen (z. B. Excel) grafisch dargestellt werden.

Für den Dateinamen wird an "Datalog" das aktuelle Datum und die Uhrzeit im ISO 8601 – Format angehängt.

```
SimpleDateFormat sdf = new SimpleDateFormat();
sdf.applyPattern("yyyyMMdd'T'HHmmss");
StringBuffer FileName = new StringBuffer().append("Datalog");
FileName = sdf.format(new Date(), FileName, new FieldPosition(0));
FileName.append(".txt");
// write to file
BufferedWriter bw = new BufferedWriter(
                 new FileWriter(FileName.toString(), true));
for (int i=0; i<length; i++) {
  bw.write((i+1) + "   " + dataLogged[i]);
  bw.newLine();
}
bw.flush();
bw.close();
}
catch (Exception e) {
  System.out.println("Exception in DatalogReceiverTest: "
                                       + e.getMessage());
}
}
}
```

Das Schreiben einer Datei erfolgt über einen **`BufferedWriter`**-Stream.

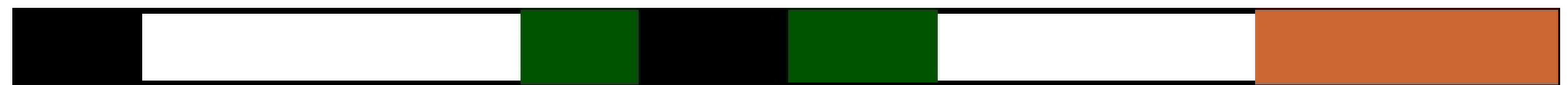
FachHochschule Offenburg
University of Applied Sciences

Graphische Darstellung der Daten

Import
in Excel

Datalog-Beispiel

Rohwert des Lichtsensors
840
820
800
780
760
740
720
700
680
0 20 40 60 80 100 120 140 160 180 200
Position

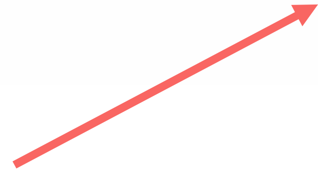


Fahrspur

Provokation einer Exception

FachHochschule Offenburg University of Applied Sciences

```
import Robots.*;.
import josx.platform.rcx.*;.
import Listeners.*;.
public class MessyBot extends TurnBot {.
  private Sensor leftTouch = Sensor.S1;.
  private Sensor rightTouch = Sensor.S3;.
  public MessyBot() {.
    leftTouch.addSensorListener(new TouchBackListener(this));
    rightTouch.addSensorListener(new TouchBackListener());.
  }.
}.
```



`TouchBackListener` wird hier kein **`TurnBot`**-Objekt übergeben.
Wird **`stateChanged()`** im **`TouchBackListener`** später aufgerufen, so zeigt **`robby`** nicht auf ein **`TurnBot`**-Objekt, sondern ist immer noch **`null`**.
Der RCX meldet auf seinem Display links vom Männchen die Zahl **`73`** und rechts davon **`8`**.

Richtige Lösung:
`TouchBackListener` sollte keinen Standardkonstruktor kodieren.

Interpretation von Exceptions

FachHochschule Offenburg University of Applied Sciences

Compileroption

```
>lejos -verbose -classpath ..;c:\Lejos\lib\classes.jar Systems.ExceptionTest
Class 0: java/lang/Object.
Class 1: java/lang/Thread.
Class 2: java/lang/String.
Class 3: java/lang/Throwable.
Class 4: java/lang/Error.
Class 5: java/lang/OutOfMemoryError.
Class 6: java/lang/NoSuchMethodError.
Class 7: java/lang/StackOverflowError.
Class 8: java/lang/NullPointerException.
Class 9: java/lang/ClassCastException.
Class 10: java/lang/ArithmeticException.
Class 11: java/lang/ArrayIndexOutOfBoundsException.
  ....
Signature 69: setPower(I)V.
Signature 70: reverseDirection()V.
Signature 71: pause(I)V.
Signature 72: <init>(LRobots/TurnBot;)V.
Signature 73: stateChanged(Ljosx/platform/rcx/Sensor;II)V.
Signature 74: <init>(I)V.
Signature 75: go_rightAngleRight()V.
Signature 76: go_rightAngleLeft()V.
Signature 77: veer_right()V.
```

Fehlerdiagnose:

8 spezifiziert die (Fehler-)klasse

-> **`NullPointerException`**

73 gibt die Nummer der Methodensignatur an

-> **`stateChanged`**

Speicherbelegung auf dem Heap

FachHochschule Offenburg University of Applied Sciences

```
import josx.platform.rcx.*;
public class StringTest {
  public static void main(String args[])
                throws InterruptedException {
    String ha = "HA";
    TextLCD.print(ha);
    Button.VIEW.waitForPressAndRelease();

    ha = ha + ' ' + ha;





    TextLCD.print(ha);
    Button.VIEW.waitForPressAndRelease();
    LCD.showNumber((int)Runtime.getRuntime().freeMemory());
    Button.VIEW.waitForPressAndRelease();
  }
}
```

```
import josx.platform.rcx.*;
public class StringBufferTest {
  public static void main(String args[])
                throws InterruptedException {
    String ha = "HA";
    TextLCD.print(ha);
    Button.VIEW.waitForPressAndRelease();

    StringBuffer bf = new StringBuffer(5);
    bf.append(ha);
    bf.append(' ');
    bf.append(ha);


    TextLCD.print(bf.toString());
    Button.VIEW.waitForPressAndRelease();
    LCD.showNumber((int)Runtime.getRuntime().freeM
    Button.VIEW.waitForPressAndRelease();
  }
}
```

Verwendung von `Strings` erzeugt viele Objekte

freier Speicher: **3924** Bytes

18 Bytes gespart

`StringBuffer`-Klasse geht pfleglicher mit dem Heap um

freier Speicher: **3942** Bytes

Effiziente Speicherbelegung auf dem Heap

FachHochschule Offenburg University of Applied Sciences

```
import josx.platform.rcx.*;
public class CharTest {
  public static void main(String args[])
              throws InterruptedException {
    char[] ha = "HA".toCharArray();
    TextLCD.print(ha);
    Button.VIEW.waitForPressAndRelease();
    char[] bf = new char[5];
    byte curpos = 0;
    for (byte i = 0; i<ha.length; i++)
      bf[curpos++] = ha[i];
    bf[curpos++] = ' ';
    for (byte i = 0; i<ha.length; i++)
      bf[curpos++] = ha[i];
    TextLCD.print(bf);
    Button.VIEW.waitForPressAndRelease();
    LCD.showNumber((int)Runtime.getRuntime().freeMemory());
    Button.VIEW.waitForPressAndRelease();
  }
}
```

Ein Feld von `chars` benötigt den geringsten Platz auf dem Heap

freier Speicher: **4080** Bytes

also **156** Bytes gespart

Da alle Programme im Speicher waren, ist der Einfluss der Programmgröße nicht berücksichtigt.

Tipps:

- Felder verwenden
- jedes neue Objekt (`new`) gut überlegen
- möglichst `byte` oder `short` verwenden
- Einsatz von tinyVM oder einem modifizierten leJOS (im Extremfall)

Zusammenfassung

FachHochschule Offenburg University of Applied Sciences

- Vorbereitungen (Roboter, leJOS, Java2) erledigt!
- Einfache Klassen mit Attributen und Methoden entworfen!
- Kontrollstrukturen (Folge, Verzweigungen, Schleifen) für Abläufe erlernt!
- Motoren und Sensoren in Betrieb genommen!
- Eine erste Platzrunde gefahren!
- Klassenhierarchie von leJOS kennenlernen.
- Schnittstellen implementieren.
- Programme "parallel" laufen lassen.
- Allerlei Nützliches zum Debuggen probieren (Sound, Display, Timer, Datalogging).