A short introduction to leJOS

LEGOMindstorms@ChairBrauer

Matthias Kranz

kranzm@in.tum.de

TU München
Introduction
What is leJOS?

- Replacement of standard firmware (see LEGO lecture)
- API that gives access to the LEGO Mindstorms hardware
- Allow programming of the LEGO Mindstorms RCX in Java (dialect)
- Implements a subset of the standard Java API
Limitations

- No garbage collection (allocated memory is not freed)
- Maximum array length of 511 entries
- No switch statement (use other constructs like if-then-else)
- No arithmetic on the long variable type
- Operator instanceof always returns true
- Implements only a subset of the standard Java API
Using leJOS
Usage of leJOS

- Environment variables: LEJOS_HOME, JAVA_HOME, RCXTTY, CLASSPATH and PATH (see values using set command; already set on the LEGO PC pool)

- lejosfirmdl (with and without -f parameter, for downloading the Java VM)

- lejosc (for compiling a lejos program to be executed on the RCX - careful, use javac for PC based lejos programs)

- lejos (for linking and downloading lejos programs to the RCX)
Programming leJOS
LCD Access I

- Three classes providing different levels of functionality (the more functionality, the more RAM is used)
- Two classes with name definitions (LCDConstants, Segment)
- MinLCD, LCD, TextLCD (in ascending order of RAM usage and functionality provided)
Usage of the three LCD classes with displaying functionality

- **MinLCD**: depiction of a number, refresh of display
- **LCD**: depiction of a number, program number or a segment and refresh
- **TextLCD**: depiction of a string or character array (of size 5), refresh
**Button Access I**

- Buttons **PRGM, VIEW, RUN** can be freely accessed, **ON–OFF** is reserved for switching the RCX on and off.

- Methods for each accessible button: `isPressed()`, `waitForPressAndRelease()` and `addButtonListener(ButtonListener bl)`

- Example: usage of `waitForPressAndRelease()` e.g. to make sure main method is not exited before all other code is executed.
Button Access II

- ButtonListener available by implementation of interface (see Java lecture, implements keyword) ButtonListener

- Methods provided by interface: 
  buttonPressed(Button b) and buttonRelease(Button b)

- Note: Works like EventListener in JAVA AWT, see JAVA AWT event model for more information
Sensor Access I

- Access depends on sensor type and mode (see LEGO lecture)
- Three static Sensor instances (one for each input port, see LEGO lecture)
- Three important sensor classes:
  - `Sensor` (for accessing the sensors)
  - `SensorConstants` (for convenient definitions of access modes and sensor types)
  - `SensorListener` (for reacting to sensor events)
Sensor Access II

Sensor types

- TOUCH, LIGHT, TEMP, ROT, RAW

Sensor modes

- ANGLE, BOOL, DEGC, DEGF, EDGE, PCT, PULSE, RAW,
Sensor functions

- activate()
- passivate()
- setTypeAndMode(int type, int mode)
- setPreviousValues(int value)
- readRawValue()
- readValue()
- readBooleanValue()
- getId()
- addSensorListener(SensorListener sl)
Sensor Access IV

- Sensor listener functions
  - `stateChanged(Sensor s, int old, int new)`
  - useful for getting notifications about sensor reading changes
- Special sensor class: `ProximitySensor`
Motor Access I

- Three static motor instances (one for each rcx output port, see LEGO lecture)
- Motor functions
  - setPower(int p), getPower(), backward(), forward(), flt(), stop(), reverseDirection(), isMoving(), isStopped(), isForward(), isBackward(), isFloating(), getId()
- Special motor class: Servo
Sound Access I

- **Two sound classes**: `Sound` and `MinSound` (descending order according to memory usage)

- **MinSound functions**
  - `playTone(int freq, int dur)`, `systemSound(boolean queued, int sound)`

- **Sound functions**
  - `beep()`, `twoBeeps()`, `beepSequence()`, `buzz`
Advanced Access

- **PersistentMemoryArea** (for storing data as long as batteries are not removed, switching on/off has no influence on stored data)
- **Memory** (only used internally)
- **Battery** (for checking battery status, important for motor speeds etc.)
- **MinuteTimer** (for resetting the internal two-byte minute-timer)
Tips, Tricks and Hints
General Notes

- Sensor values and motor speeds directly depend on the accumulator charge.
- Light sensor values are highly influenced by surrounding light conditions.
- IR communication suffers under high light conditions, data may be loss or data transfer slow (depending on the transfer protocol).
No instances for hardware (e.g. sensors), static access

Network access requires instance of correct protocol

Be careful when using strings or loop variable etc. (no garbage collection)
Programming Notes II

- Use bytes and shots where possible (instead of ints and long)
- Re-use variables (e.g. loop counters, declare them once at the beginning and (re-)use them as often as possible)
- More generally: Do not allocate objects inside loops
- Be careful with Strings - use StringBuffers!
Check whether the Min-Classes are sufficient for your needs and if you really need to use the big versions of the access classes (e.g. is MinSound sufficient enough for debugging outputs?)

Check memory usage regularly!

Make use of the main method (do not pack your work in other threads and make the main method just make sure the program is not terminated before time)
Bibliography
References


End
[FGH+02] as [CW98] as a [LJ:] as