

# Mindroid Pilotworkshop Aufgabenstellung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

11. bis 13.06.2018

Fachgebiet Echtzeitsysteme / MAKI  
TU Darmstadt

---

## Aufgabe 1 Hallo Welt

---

In der Informatik ist es üblich, ein Hallo-Welt-Programm<sup>1</sup> zu schreiben, wenn man eine Programmierungsumgebung kennenlernt. Deshalb fangen wir damit an.

---

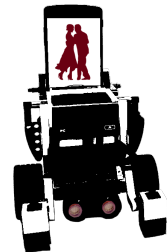
### Aufgabe 1.1 Setup

---

Damit alles funktioniert, musst du deine Geräte zuerst überprüfen und einrichten. Dabei lernst du, wie du die einzelnen Komponenten miteinander verbindest und neu erstellte Programme auf das Smartphone überträgst. Zuerst musst du dazu das Smartphone und den Roboter mit dem **USB-Kabel** verbinden. Überprüfe dabei ob

- Alle Kabel am Roboter ordentlich sitzen und
- Dein Smartphone entweder mit dem WLAN **MD005-2.4GHz** oder **MD0029-2.4GHz** verbunden ist.

Nun kannst du den Roboter durch Drücken der Enter-Taste starten. Die Tastenbelegung kannst du im Anhang in Abbildung A.1 auf Seite 18 nachschauen. Während der Roboter startet, kannst du auch die Mindroid-App auf dem Handy starten. Du musst nicht warten bis der Roboter fertig gestartet ist.



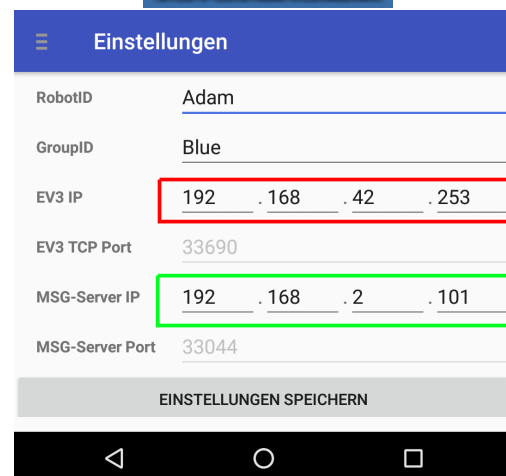
---

#### Aufgabe 1.1.1 App und PC verbinden

---

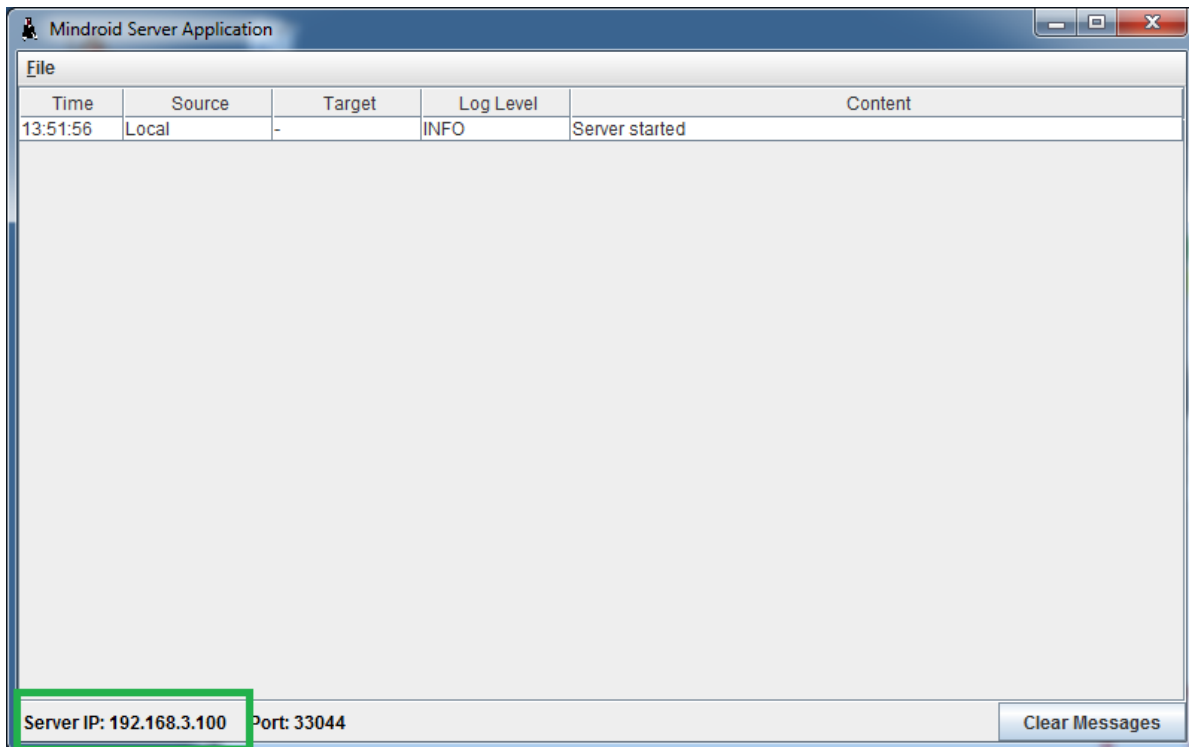
Um neue Programme einfach und schnell auf das Smartphone übertragen zu können, muss zuerst das Smartphone mit dem PC verbunden werden. Dazu sind folgende Schritte nötig:

1. Starte den **Message-Server**, indem du die **ServerStarten.bat** auf dem Desktop des Computers mit einem Doppelklick ausführst. Der Server erledigt ein paar Einstellungen. Außerdem ist er später wichtig, damit die Roboter untereinander kommunizieren können.
2. Um eine dauerhafte Verbindung herzustellen, müssen wir nun das Smartphone mit dem Server bekannt machen.
3. Der Message-Server zeigt dir links unten in der Ecke seine IP-Adresse an (siehe Screenshot auf Seite 3). Diese musst du nun der Mindroid-App mitteilen. Dazu navigierst du über das Menü oben links in den **Einstellungen-Bildschirm** (Abbildung rechts). Dort trägst du unter **MSG-Server IP** die IP des Servers ein. Ist dies erledigt, kannst du zurück zum Hauptbildschirm der App und auf **Verbinden** klicken. Die App verbindet sich nun mit dem Server.



---

<sup>1</sup> <https://de.wikipedia.org/wiki/Hallo-Welt-Programm>

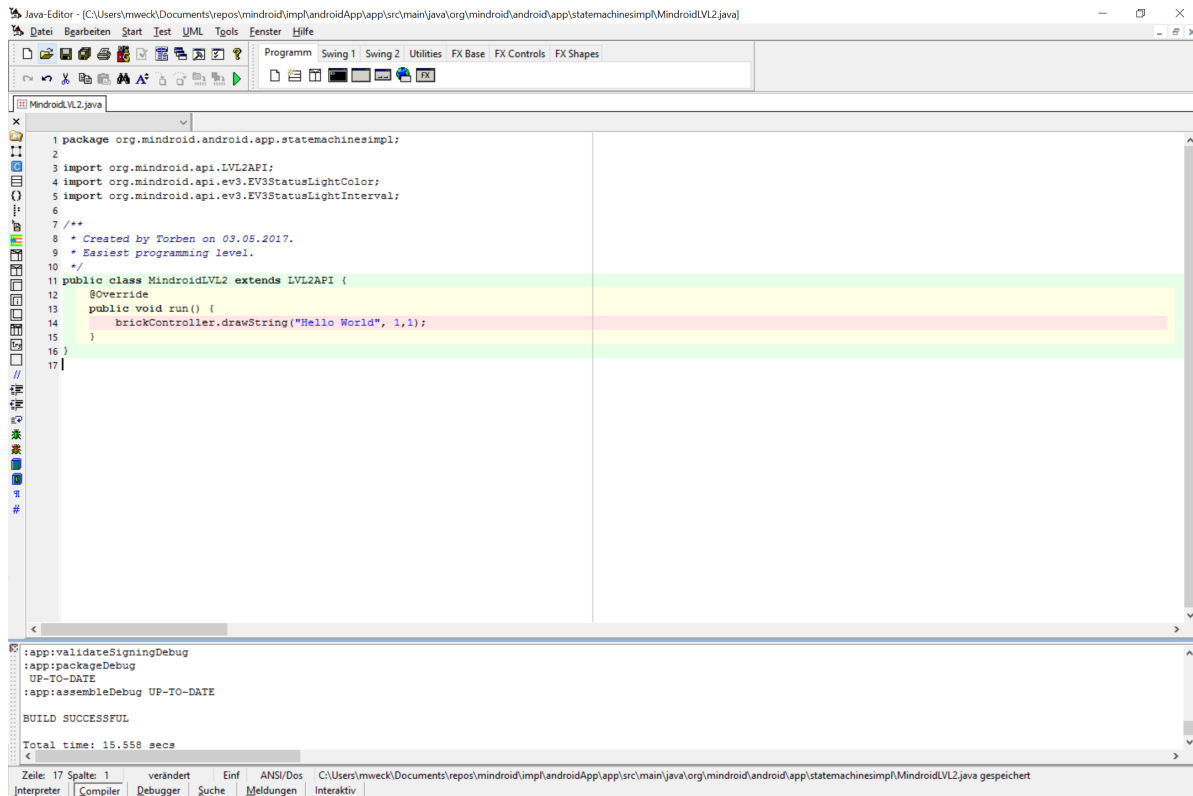


**Abbildung Aufgabe 1.1:** Screenshot des Servers


## Aufgabe 1.1.2 Programme übertragen

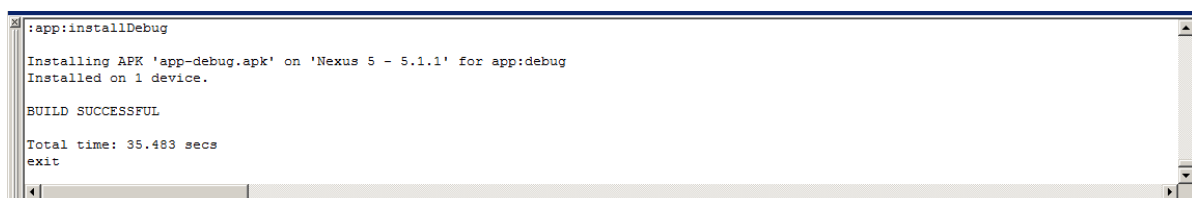
Neue Programme schreiben und übertragen wir in der Entwicklungsumgebung "**Java-Editor**".

1. Starte diesen indem du die Verknüpfung auf dem Desktop startest. In der Mitte findest du den Quelltext des aktuellen Programms, hinterlegt mit Farben, die die Orientierung erleichtern sollen.



Wie das Programm genau funktioniert, sehen wir uns gleich an. Zunächst einmal wollen wir es auf das Smartphone spielen, um zu sehen, was passiert.

2. Dazu klickst du auf den blauen Play-Button oben links (Der Button ändert seine Farbe nach dem Durchlauf wieder zu türkis).  Dein "Hallo Welt"-Programm wird jetzt an das Handy übertragen.
3. In der Konsolenausgabe am unteren Bildrand sollte in etwa folgender Text erscheinen: Das



Programm ist nun übertragen.

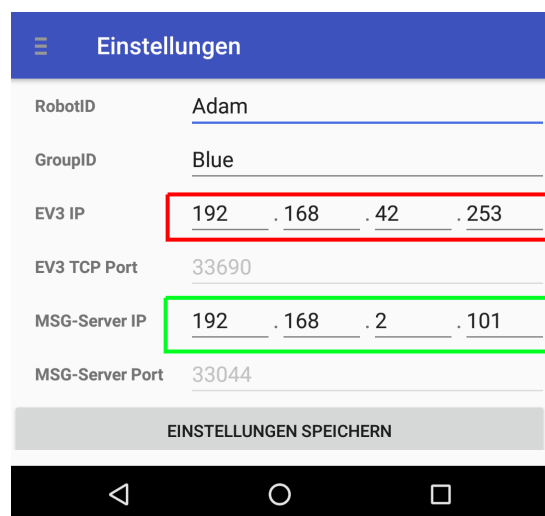
### Aufgabe 1.1.3 Roboter und App verbinden

Zuletzt müssen nun noch Smartphone und Roboter miteinander verbunden werden.

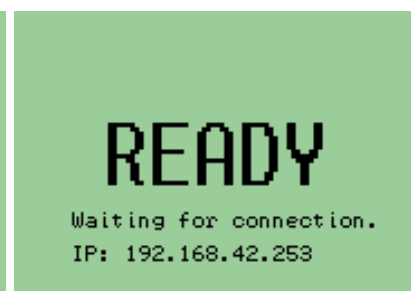
1. Überprüfe dazu, ob wie rechts gezeigt die IP-Adresse 192.168.42.253 im Hauptmenü angezeigt wird. Ist dies nicht der Fall, folge der Anleitung in Abschnitt B auf Seite 19.



2. Wird die **IP-Adresse** im Hauptmenü des Roboters richtig angezeigt, musst du nun noch kontrollieren, ob diese IP-Adresse auch im Einstellungsmenü der App richtig gesetzt ist. Überprüfe dazu den Eintrag unter **EV3 IP**. Die farblichen Markierungen auf den Bildern sind dir dabei eine Hilfe. Falls der Eintrag nicht korrekt ist, korrigiere ihn.



3. Nun kannst du das **EV3-Programm** starten. Wähle dazu **Run Default** im Hauptmenü des Roboters aus und bestätige mit der Auswahlstaste. Danach musst du warten, bis der Roboter "Ready" anzeigt



4. Der Roboter ist nun bereit dazu sich mit dem Handy zu verbinden. In der App kannst du nun „Verbindung zum EV3-Brick herstellen“ antippen.
5. Um nun das Hello-World-Programm zu starten, wähle es in der Liste der App aus und drücke auf **Starten**. Es wird ausgeführt, und auf dem Bildschirm des Roboters erscheint "Hallo Welt!"

---

## Aufgabe 1.2 Der Quelltext erklärt

---

Nachdem du den Roboter erfolgreich eingerichtet und das erste Mal getestet hast, gehen wir jetzt daran, uns den Quelltext näher anzusehen.

```
1 import org.mindroid.api.ImperativeWorkshopAPI;
2 import org.mindroid.impl.brick.Textsize;
3
4 public class HelloWorld extends ImperativeWorkshopAPI {
5
6     public HelloWorld() {
7         super("Hello World");
8     }
9
10    @Override
11    public void run() {
12        clearDisplay();
13        drawString("Hello World", Textsize.MEDIUM, 10 , 50);
14    }
15 }
```

Das Verhalten des Roboters befindet sich in der **run-Methode** (Zeilen 13-16). Wenn ein Mindroid-Programm ausgeführt wird, werden die Befehle in dieser Methode nacheinander abgearbeitet.

- **clearDisplay()** löscht den Display-Inhalt
- **drawString(text, textsize, xPosition, yPosition)** schreibt einen gegebenen Text (**text**) an die gegebenen Koordinaten (**xPosition, yPosition**) und verwendet dabei die definierte Schriftgröße (**textsize**).
- Der Konstruktor in den Zeilen 8 bis 10 gibt unserem Programm einen Namen (Zeile 8). Mit dem Aufruf von **super("Hello World")** bestimmen wir, unter welcher Bezeichnung unser Programm später ausgewählt werden kann. Es ist sinnvoll an beiden Stellen aussagekräftige Bezeichnungen zu verwenden.

Daneben gibt es noch die sogenannten **“imports”**. Da die Programm-Bibliotheken der MindroidApp sehr groß sind, hat jede Klasse einen ausführlichen Namen, der dabei hilft, den Überblick zu bewahren. Der Teil vor dem Klassennamen heißt **Paket** (engl. package). Zum Beispiel ist die Klasse **ImperativeWorkshopAPI** im Paket **org.mindroid.api**.

---

## Aufgabe 1.3 Erste eigene Änderungen vornehmen

---

Um zu sehen, wie die Entwicklung deiner eigenen Roboter-Software im Folgenden ablaufen wird, wirst du nun am Hello-World-Programm eine kleine Änderung durchführen und anschließend das aktualisierte Programm auf den Roboter laden.

Öffne dazu zuerst die **HelloDate.java**-Datei. Ändere den Code nun, sodass er dem folgenden Code ähnelt:

**Wichtig:** Ändere in keiner Aufgabe die imports oder den Package-Namen. Diese sind bereits so wie sie sein müssen.

```
1 import org.mindroid.api.ImperativeWorkshopAPI;
2 import org.mindroid.impl.brick.Textsize;
```

```

3 import java.text.SimpleDateFormat;
4 import java.util.Date;
5
6 public class HelloDate extends ImperativeWorkshopAPI {
7
8     public HelloDate() {
9         super("Hello Date");
10    }
11
12    @Override
13    public void run() {
14        SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyy");
15        clearDisplay();
16        drawString("Datum: " + formatter.format(new Date()), Textsize.MEDIUM,
17                  ↪ 10, 50);
18    }
19 }

```

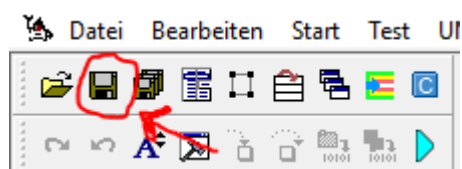
- Die Imports von **java.text.SimpleDateFormat** und **java.util.Date** sind neu hinzugekommen.
- Außerdem gibt es nun eine Variable **formatter** (Zeile 17), die in der Lage ist, das aktuelle Datum formatiert auszugeben (**formatter.format(new Date())**, Zeile 19)).

Um die Änderungen nun auf das Handy zu übertragen, muss die App neu installiert werden.

1. Zuerst musst du deine Änderungen am Code speichern<sup>2</sup>.
2. Betätige jetzt die Schaltfläche “Starten”.
3. Nachdem im Compiler-Fenster die Meldung “BUILD SUCCESSFUL” erschienen ist, wird auf dem Handy die Mindroid-App automatisch geschlossen und wieder geöffnet.
4. Wähle “Verbindung zum EV3-Brick herstellen”
5. Wähle im Dropdown “HelloWorld” aus und betätige “Start”.
6. Auf dem Display sollte nun das aktuelle Datum ausgegeben werden.

### Übrigens

Falls du erfahren möchtest, wie man bspw. die aktuelle Uhrzeit mithilfe von SimpleDateFormat ausgibt, klicke mit Rechts auf den Klassennamen SimpleDateFormat und wähle den Eintrag “API-Hilfe.” Im sich öffnenden Fenster findest du die Dokumentation der Formatierungsparameter, die im Konstruktor verwendet werden. Die vollständige Doku aller Klassen für die Mindroid-App erhältst du, indem du auf dem Desktop den Link “Mindroid Doku” anklickst. Die vollständige Doku aller Klassen der Java-Standardbibliothek erhältst du, indem du auf dem Desktop den Link “JDK Doku” anklickst oder die folgende URL aufrufst: <https://docs.oracle.com/javase/7/docs/api/index.html>



<sup>2</sup> Oben Links auf das Diskettensymbol klicken

---

## Aufgabe 2 Den Roboter kennenlernen

---

In dieser Aufgabe wollen wir gemeinsam die Fähigkeiten des Roboters kennenlernen. Das Display ist nützlich, um bestimmte Informationen schnell auszugeben, aber wirklich nützlich wird der Roboter erst durch seine beiden **Antriebsmotoren**, den **Gyrosensor**, die beiden **Lichtsensoren** und den **Ultraschallsensor**.

---

### Aufgabe 2.1 Fahren - die Antriebsmotoren

---

Wir schauen uns als Erstes an, wie man den Roboter fahren lassen kann. Der Roboter unterstützt folgende **Bewegungsrichtungen**:

- Vorwärts fahren,
- Rückwärts fahren,
- Drehen nach Rechts und
- Drehen nach Links

Um die Fahr-Fähigkeiten des Roboters kennenzulernen, wirst du den folgenden Code nachprogrammieren, der ein etwas **spezielles Quadrat** fährt.

```
1  import org.mindroid.api.ImperativeWorkshopAPI;
2
3  public class DriveSquare extends ImperativeWorkshopAPI {
4
5      public DriveSquare() {
6          super("Drive Square");
7      }
8
9      @Override
10     public void run() {
11         for (int i = 0; i < 3 && !isInterrupted(); i++) {
12             int angle = 90;
13             forward();
14             delay(1000);
15             turnRight(angle);
16             forward();
17             delay(1000);
18             turnLeft(angle);
19             backward();
20             delay(1000);
21             turnRight(angle);
22             backward();
23             delay(1000);
24             turnLeft(angle);
25         } // end of for
26         stop();
27     }
28 }
```

- Alle benötigten Methoden werden von der Klasse **ImperativeWorkshopAPI** bereitgestellt. Wir brauchen daher keine zusätzlichen Imports.



- In Zeile 14 bestimmt die **for-Schleife**, dass der Roboter das Quadrat dreimal abfahren soll.
- Der **forward()**-Befehl (Zeile 16) setzt den Roboter in Bewegung. Dabei fährt der Roboter solange, bis er einen anderslautenden Befehl erhält (wie bspw. **stop()** in Zeile 29).
- Anschließend wird für 1 Sekunden **gewartet** (Zeile 16). In dieser Zeit fährt der Roboter vorwärts.
- Daraufhin vollführt der Roboter eine Drehung nach Rechts um 90° (**turnRight(angle)**, Zeile 18).
- Nach der nächsten Geraden (Zeile 19) dreht der Roboter nach links (**turnLeft(angle)**, Zeile 21), um anschließend die verbleibenden beiden Geraden rückwärts zu fahren (**backward()**-Befehl, Zeile 22 und 25).

---

## Aufgabe 2.2 Der Ultraschallsensor - Abstand Messen

---

Um dich mit dem Ultraschall-Distanzssensor vertraut zu machen, implementiere den folgenden Code nach. Er stellt einen einfachen Parksensor dar, der wie folgt funktioniert:

1. Liegt die Distanz zu einem Objekt vor dem Roboter **unter 15cm**, dann blinkt die Status-LED schnell rot und es wird **“Oh oh :-O”** ausgegeben.
2. Liegt die Distanz **zwischen 15cm und 30cm**, dann blinkt die Status-LED gelb und es wird **“Hm :-/”** ausgegeben.
3. Liegt die Distanz **über 30cm**, dann leuchtet die Status-LED grün und es wird **“OK :-)”** ausgegeben.

```

1  import org.mindroid.api.ImperativeWorkshopAPI;
2  import org.mindroid.api.ev3.EV3StatusLightColor;
3  import org.mindroid.api.ev3.EV3StatusLightInterval;
4  import org.mindroid.impl.brick.Textsize;
5
6  public class ParkingSensor extends ImperativeWorkshopAPI {
7
8      public ParkingSensor() {
9          super("Parking Sensor");
10     }
11
12     @Override
13     public void run() {
14         String previousState = "";
15         clearDisplay();
16         drawString("Parking sensor", Textsize.MEDIUM, 10, 10);
17         while (!isInterrupted()) {
18             clearDisplay();
19             if (getDistance() < 0.30f && getDistance() > 0.15f) {
20                 drawString("Hm :-/", Textsize.MEDIUM, 10, 10);
21                 if (!previousState.equals("hm")) {
22                     setLED(EV3StatusLightColor.YELLOW,
23                         EV3StatusLightInterval.BLINKING);
24                 }
25                 previousState = "hm";
26             } else if (getDistance() < 0.15f) {

```

```

27         drawString("Oh oh :-0", Textsize.MEDIUM, 10, 10);
28         if (!previousState.equals("oh")) {
29             setLED(EV3StatusLightColor.RED,
30                 EV3StatusLightInterval.DOUBLE_BLINKING);
31         }
32         previousState = "oh";
33     } else {
34         drawString("OK :-)", Textsize.MEDIUM, 10, 10);
35         if (!previousState.equals("ok")) {
36             setLED(EV3StatusLightColor.GREEN,
37                 EV3StatusLightInterval.ON);
38         }
39         previousState = "ok";
40     }
41     delay(100);
42 }
43 }
44 }

```

- Um die LED ansteuern zu können, müssen wir die Pakete **org.mindroid.api.ev3.EV3StatusLightColor** und **org.mindroid.api.ev3.EV3StatusLightInterval** importieren.
- Wie in Zeile 19 zu sehen ist, läuft das Programm in einer Endlosschleife, bis der “Stop”-Knopf in der App betätigt wird.
- Wir müssen uns jeweils den vorherigen Zustand in der Variablen **previousState** (Zeile 17) merken, da wir ansonsten alle 100ms den Zustand der LED zurücksetzen würden, was das Blinken verhindert. Mithilfe von **previousState** ändern wir den LED-Modus nur dann, wenn wir müssen.

---

### Aufgabe 2.3 Die Farbsensoren - Farbe messen

---

In dieser Aufgabe lernst du die Farbsensoren des Roboters kennen. Der folgende Quelltext liest kontinuierlich den aktuell gemessenen Farbwert des linken und rechten Lichtsensors aus (**getLeftColor()** bzw. **getRightColor()** in Zeilen 16 und 17).

```

1  import org.mindroid.api.ImperativeWorkshopAPI;
2  import org.mindroid.impl.brick.Textsize;
3  import org.mindroid.impl.statemachine.properties.Colors;
4
5  public class ColorTest extends ImperativeWorkshopAPI {
6
7      public ColorTest() {
8          super("Color Test");
9      }
10
11     @Override
12     public void run() {
13         while (!isInterrupted()) {
14             Colors leftColorValue = getLeftColor();

```

```

15         Colors rightColorValue = getRightColor();
16
17         clearDisplay();
18         drawString("Colors", Textsize.MEDIUM, 1, 1);
19         drawString("L: " + describeColor(leftColorValue), Textsize.MEDIUM, 1,
20             ↳ 17);
21         drawString("R: " + describeColor(rightColorValue), Textsize.MEDIUM, 1,
22             ↳ 33);
23         drawString("Distance: " + getDistance(), Textsize.MEDIUM, 1, 51);
24         delay(500);
25     }
26 }
27
28 private static String describeColor(final Colors colorValue) {
29     if (colorValue == Colors.NONE) return "None";
30     if (colorValue == Colors.BLACK) return "Black";
31     if (colorValue == Colors.BLUE) return "Blue";
32     if (colorValue == Colors.GREEN) return "Green";
33     if (colorValue == Colors.YELLOW) return "Yellow";
34     if (colorValue == Colors.RED) return "Red";
35     if (colorValue == Colors.WHITE) return "White";
36     if (colorValue == Colors.BROWN) return "Brown";
37     return "unknown";
38 }
39 }

```

- Die Methode **describeColor** (Zeilen 29-39) zeigt, wie du den Rückgabewert in einen lesbaren Text umwandelst.
- In den Zeilen 20-21 siehst du, wie man auf dem Display mehrzeiligen Text ausgeben kann. Die Buchstaben haben jeweils eine Höhe von 16 Pixeln, sodass die zweite Zeile an der y-Position 17 und die dritte Zeile an der y-Position 33 beginnt.
- Um die Qualität der Farbmessung näher zu betrachten, haben wir für dich Farbtafeln mit allen sieben unterstützten Farben des EV3-Lichtsensors vorbereitet. Bei welchen Farben funktioniert die Erkennung gut, bei welchen eher weniger?
- Der Farbsensor kann auch zur Erkennung von Abgründen eingesetzt werden: Welche Farbwerte werden gemessen, wenn der Roboter auf der Tischplatte steht und wenn die Farbsensoren über den Tischrand ragen?

---

## Aufgabe 2.4 Kommunikation zwischen Robotern

---

In der vorherigen Aufgaben hast du kennengelernt, wie ein Programm auf einem einzelnen Roboter ausgeführt wird. Als nächstes wollen wir die Roboter **miteinander sprechen lassen**.

Auch hier starten wir mit einem einfachen (diesmal verteilten) “Hallo Welt!”-Programm. Die Kommunikation läuft über das bereits vorgestellten “Server”-Programm, welches ihr vorhin schon auf dem Entwicklungsrechner gestartet habt.

Damit die Roboter voneinander unterschieden werden können, benötigt jeder einen eigenen Namen. Um diese Einstellungen ändern zu können, müsst ihr die Verbindung zum Server erst einmal trennen. Navigiert nun wieder in das Einstellungs-Menü der App und gebt den Robotern Namen. Stellt sicher, dass die Roboter auch in Gruppen eingeteilt sind.

---

Wiederhole diesen Schritt nun auch für den zweiten Roboter. In unserem Beispiel gehen wir davon aus, die Roboter heißen Robert und Berta.

Wir möchten nun, dass Berta eine Nachricht mit dem Inhalt **“Hallo Robert!”** an den Nachrichtenserver versendet. Robert soll diese Nachricht empfangen und die Nachricht auf seinem Display ausgeben. Dazu sind zwei unterschiedliche Programme für Robert und Berta notwendig.

```
1 import org.mindroid.api.ImperativeWorkshopAPI;
2 import org.mindroid.impl.brick.Button;
3
4 public class HelloWorldPingA extends ImperativeWorkshopAPI {
5
6     public HelloWorldPingA() {
7         super("Hello World Ping A");
8     }
9
10    @Override
11    public void run() {
12        clearDisplay();
13        while(!isInterrupted()){
14            delay(10);
15            if(isButtonClicked(Button.ENTER))
16                sendMessage("Robert", "Hallo Robert!");
17        }
18    }
19 }
```

- Bei Programmstart sendet Berta in Zeile 16 eine Nachricht an **Robert** mit den Inhalt **“Hallo Robert!”**

```
1 import org.mindroid.api.ImperativeWorkshopAPI;
2 import org.mindroid.impl.brick.Textsize;
3
4 public class HelloWorldPingB extends ImperativeWorkshopAPI {
5
6     public HelloWorldPingB() {
7         super("Hello World Ping B");
8     }
9
10    @Override
11    public void run() {
12        clearDisplay();
13        while(!isInterrupted()){
14            if (hasMessage()){
15                String msg = getNextMessage().getContent();
16                if (msg.equals("Hallo Robert!")){
17                    drawString("Nachricht von Berta erhalten", Textsize.MEDIUM, 1,
18                               ↪ 60);
19                }
20                delay(100);
21            };
22        }
23    }
```

- Robert überprüft mit **hasMessage()** (Zeile 16) ob neue Nachrichten auf dem Message-Server vorhanden sind.
- Sobald eine Nachricht vorliegt, wird der Inhalt der Nachricht in die Variable **msg** gespeichert (Zeile 16).
- die Nachricht wird nun mit dem String **“Hallo Robert!”** verglichen<sup>3</sup>. Stimmen beide überein, schreibt Robert auf sein Display einen Text (Zeile 19).

---

## Aufgabe 2.5 Nächste Schritte

---

Du hast jetzt alle wesentlichen Fähigkeiten des Roboters kennengelernt. Nun ist es an der Zeit, dass du deine eigenen Projekte umsetzt. Die folgenden Aufgaben geben dir dazu Anregungen. Falls du bestimmte Sensorwerte zur Kalibrierung nutzen willst, ist dir die Anwendung Sensor Monitoring bestimmt eine Hilfe.

1. Wähle dazu auf der Übersichtsseite des der Mindroid-App im Dropdown **“SensorMonitoring”** aus.
2. Betätige **“Start”**.
3. Öffne anschließend im Menü links oben den Eintrag **“Sensor Monitoring”**.
4. Du siehst die aktuell gemessenen Werte der Sensoren
  - Der Wert **“Color ID”** beschreibt die aktuell gemessene Farbe unter dem linken Sensor. Ihre Werte sind in der Klasse

**org.mindroid.impl.statemachine.properties.sensorproperties.Color**

dokumentiert. Wenn du ein weißes Blatt Papier direkt unter den linken Farbsensor legst, sollte der Wert auf 6 (=Weiß) wechseln (manchmal wird auch 2 (=Blau) gemessen). Wenn du einen schwarzen Gegenstand direkt unter den Sensor legst, sollte der Wert auf 1 wechseln. Die Farbbestimmung mittels Color-ID ist leider relativ unpräzise.

- Der Wert **“Distance”** gibt die aktuell vom Ultraschallsensor gemessene Distanz in Metern an. Bewege deine Hand vor dem Sensor vor und zurück und beobachte die Veränderung des Wertes.
- Der Wert **“Angle”** gibt den aktuell gemessenen Winkel des Gyro-Sensors an. Drehe den Roboter um die Hochachse und beobachte die Veränderung des Wertes.

---

## Aufgabe 2.6 Nützliche Imports

---

- *import org.mindroid.api.ImperativeWorkshopAPI* für die Elternklasse
- EV3-Statuslicht:
  - Farbe: *import org.mindroid.api.ev3.EV3StatusLightColor;*
  - Werte: **EV3StatusLightColor.GREEN, .RED, .YELLOW, .OFF**
  - Blinkhäufigkeit: *import org.mindroid.api.ev3.EV3StatusLightInterval;*
  - Werte: **EV3StatusLightInterval.ON, .BLINKING, .DOUBLE\_BLINKING**

---

<sup>3</sup> Beachte: Strings werden in Java nicht mit == verglichen, sondern mittels der **equals()**-Methode

---

### Aufgabe 3 Wand-Ping-Pong

---

Nutze deine Kenntnisse, um den Roboter wie einen Ping-Pong-Ball in gerader Linie zwischen zwei Wänden/Gegenständen/... hin und her fahren zu lassen. Anweisungen:

1. Der Roboter soll solange geradeaus fahren, bis er eine Wand erkennt. Er soll dabei so nah wie möglich an die Wand heranhfahren, ohne mit ihr zu kollidieren.
2. Dann soll er ein kleines Stück rückwärts fahren und sich um  $180^\circ$  drehen
3. Beginne bei 1.

Bewertungskriterien:

- (1P) Ist die  $180^\circ$ -Drehung exakt?
- (1P) Vergleich: Wie nahe kommst du an die Wand heran?
- (1P) Vergleich: Wie schnell schaffst du eine Runde (Fahren  $\rightarrow$  Wand+Drehen  $\rightarrow$  Fahren)?

---

### Aufgabe 4 Koordiniertes Wand-Ping-Pong

---

Diese Aufgabe lehnt sich an die vorherige an. Allerdings sollen sich nun zwei Roboter abstimmen. Beide Roboter starten nebeneinander und blicken in die gleiche Richtung. Anweisungen:

1. Roboter A fährt solange, bis er eine Wand entdeckt. Er setzt zurück, dreht sich um und bleibt stehen.
2. Roboter A sendet Roboter B eine "Start"-Nachricht.
3. Daraufhin setzt sich Roboter B in Bewegung, bis er auf die Wand trifft. Daraufhin setzt Roboter B zurück, wendet und bleibt stehen.
4. Roboter B sendet Roboter A eine "Weiter"-Nachricht.
5. Beginne bei 1.

Bewertungskriterien:

- (1P) Roboter kollidieren nicht.
- (1P) Vergleich: Wie schnell schaffen die Roboter eine "Runde" (A: Fahren  $\rightarrow$  Wand + Drehen + Stop; B: Warten  $\rightarrow$  Fahren  $\rightarrow$  Wand + Drehen + Stop)? Hier sollen die Roboter nicht möglichst nahe, sondern nahe genug an die Wand heranhfahren (Abstand der Vorderachse kleiner 20cm).



---

## Aufgabe 5 Mähroboter

---



Nutze deine Kenntnisse, um den Roboter in einem mit schwarzem (oder weißem) Klebeband abgesperrten Bereich herumfahren zu lassen (so ähnlich wie beispielsweise viele Mähroboter arbeiten).

1. Wie beim Wand Ping-Pong soll der Roboter erstmal geradeaus fahren.
2. Wenn er eine Grenze erkennt, soll er zurücksetzen und sich eine neue Richtung aussuchen.
3. Beginne bei 1.

Tipp: Überprüfe, welche Color-IDs auf dem Boden und den Begrenzungen erkannt werden, um festzustellen, wann der Roboter an die Umzäunung gelangt ist.

Bewertungskriterien:

- (1P) Der umgrenzte Bereich darf nicht verlassen werden! Keines der Räder darf die Umgrenzung berühren!
- (1P) Vergleich: Wer schafft es am schnellsten, alle vier Seiten eines viereckigen Bereichs zu “treffen”?

---

## Aufgabe 6 Platooning

---



In dieser Aufgabe geht es darum, zwei Roboter hintereinander her fahren zu lassen, ohne dass es einen Auffahrunfall gibt. Aktuell forschen zahlreiche Unis und Unternehmen unter dem Schlagwort Platooning an genau dieser Problemstellung bei echten LKWs und PKWs: Die Fahrzeuge fahren dabei so nahe, dass sie den Windschatten des Vorfahrenden ausnutzen können.

Roboter A und B werden hintereinander platziert, sodass sie in die gleiche Richtung blicken. Ziel ist es zunächst, dass Roboter B den Abstand zu Roboter A in einem bestimmten Toleranzbereich hält. Die Distanzangaben im Folgenden sind nur mögliche Werte - du bestimmst selbst, was geeignete Grenzwerte sind.

1. Roboter A fährt los. Sobald der Abstand zwischen Roboter A und Roboter B größer als 35cm wird, beginnt Roboter B aufzuschließen.
2. Wird der Abstand kleiner als 25cm, hält Roboter B die Geschwindigkeit von Roboter A .
3. Wird der Abstand kleiner als 15cm, lässt Roboter B sich zurückfallen (oder fährt sogar rückwärts).

### Bewertungskriterien

- (1P) Roboter A und B fahren mind. 1 Meter hintereinander her ohne Kollisionen (weder mit anderen Robotern noch mit der Wand).
- (1P) Abstand bleibt in einem von euch zuvor bestimmten Bereich (Schnur), wenn beide Roboter vorwärts fahren.
- (1P) Vergleich: Geringstmöglicher Abstand, bei dem keine Kollision stattfindet.



---

## Aufgabe 7 Dancing Robots

---



Beim Cha-Cha-Cha gibt es die Tanzfigur “Verfolgung”. Dabei verfolgt jeweils ein Tanzpartner den anderen, bis beide sich umdrehen und die Rollen wechseln. Diese Figur ist tatsächlich nicht sehr weit vom Platooning-Beispiel aus der vorherigen Aufgabe entfernt. Der Ablauf soll dieses Mal wie folgt aussehen:

1. Roboter A übernimmt zunächst das Kommando und fährt voraus, während Roboter B einen möglichst gleichbleibenden Abstand hält.
2. Roboter A beschließt nach einer gewissen Zeit, dass nun die Drehung folgt. Er stoppt und sendet eine “Drehen”-Nachricht an Roboter B.
3. Roboter B stoppt, wendet und sendet Roboter A eine “Gedreht”-Nachricht.
4. Daraufhin dreht Roboter A ebenfalls um  $180^\circ$ .
5. Nun tauschen Roboter A und B die Rollen: Roboter B fährt voraus und gibt den Ton bis zur nächsten Drehung an.

### Bewertungskriterien

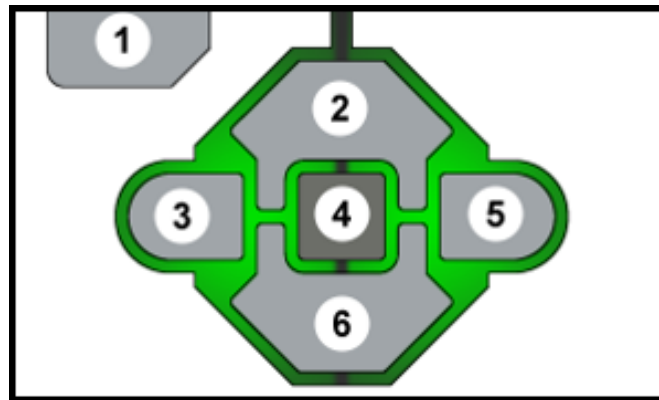
- (1P) Einmal Verfolgung hin (Roboter A ist der Führende) und einmal Verfolgung zurück (Roboter B ist der Führende) und dabei keine Kollision!
- (1P) Es sollte klar erkennbar sein (bspw. per LED-Statusleuchte), wer aktuell der “Führende” ist.

---

## A EV3 Tasten

---

Abbildung A.1 zeigt dir wie die Tasten am EV3-Brick genannt werden. Die Enter-Taste wird zum Bestätigen genutzt, mit der Escape-Taste, geht es ein Menü zurück.



**Abbildung A.1:** EV3-Tastenbelegung<sup>4</sup>

Die Bedeutung der Tasten kannst du der folgenden Aufzählung entnehmen.

1. **Escape / Zurück**
2. **Up / Hoch**
3. **Left / Links**
4. **Enter / Bestätigen**
5. **Right / Rechts**
6. **Down / Unten**

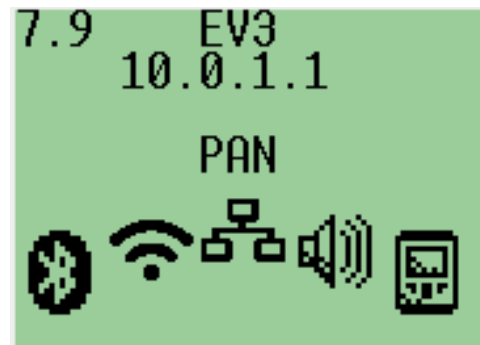
---

<sup>4</sup> Quelle <http://www.ev3dev.org/images/ev3/labeled-buttons.png>

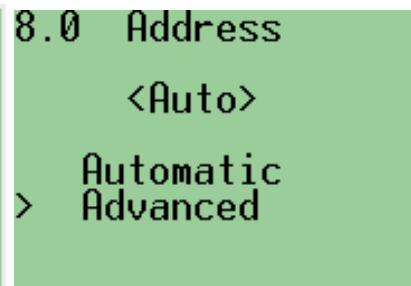
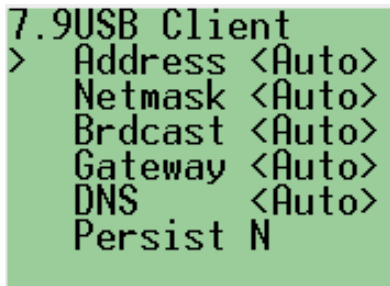
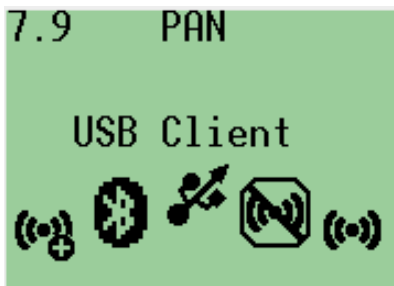
## B PAN Einrichtung

Wird im Hauptmenü noch nicht die richtige IP-Adresse angezeigt, müssen zuerst die PAN<sup>5</sup>-Einstellungen korrigiert werden.

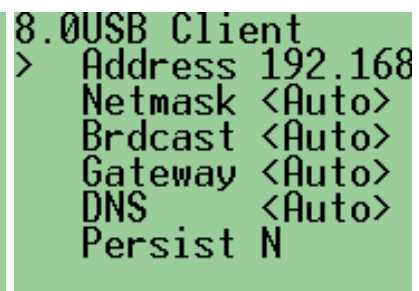
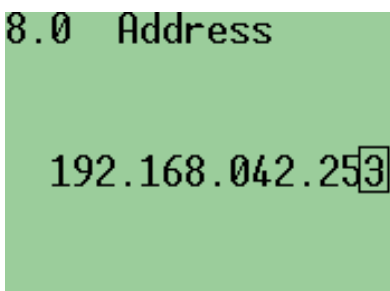
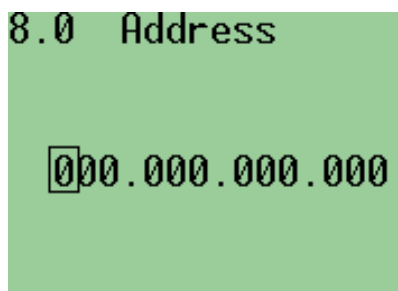
1. Dazu musst du zuerst in das **PAN-Menü** des Roboters navigieren. Wechsle mit den Links-/Rechts-Tasten bis du den Menüpunkt **PAN** siehst und betätige die **Auswahltaste**.



2. Nun navigierst du durch das Menü des Roboters wie auf den Bildern zu sehen und bestätigst jeweils mit der Auswahltaste: **USB-Client - Address - Advanced**



Nun musst du die IP Adresse 192.168.42.253 einstellen. Dazu navigierst du mit den rechts-/links-Tasten zu den einzelnen Ziffern und änderst deren Wert mit den oben-/unten-Tasten. Orientiere dich an den Bildern! Am Ende bestätigst du wieder mit der Enter-Taste.



3. Mit der **Zurück**-Taste kommst du wieder in das Hauptmenü und die Einstellungen werden übernommen.



Nun kannst du wieder zu Punkt 3 auf Seite 5 wechseln.

<sup>5</sup> PAN = Personal Area Network

---

## C Troubleshooting

---

### C.1 Installation über WLAN funktioniert nicht

---

Im Message-Server über **File->Connected Devices** schauen ob alle Smartphones in der Liste auftauchen und der ADB-state auf **connected** steht. Ist dies nicht der Fall, tippe in der App auf **TRENNEN** und stelle die Verbindung danach erneut her. Falls das nicht klappt, kontaktiere einen Betreuer. Falls das Installieren per WLAN gar nicht mehr funktioniert, kann jederzeit eine USB-Verbindung zwischen Smartphone und PC hergestellt werden und darüber die App installiert werden.

---

## D Sensorbelegung

---

Tabelle D.1 zeigt die standardmäßige Sensorbelegung, wie sie in der App unter “Mein Roboter” definiert sein muss.

Anschluss	Sensortyp	Modus
1	Farbe	ColorID
2	Ultraschall	Distance
3	Gyroskop	Angle
4	Farbe	ColorID

**Tabelle D.1:** Sensorbelegung

Tabelle D.2 zeigt den standardmäßigen Motoranschluss, wie er in der App unter “Mein Roboter” definiert sein muss.

Anschluss	Motor
A	Large Regulated Motor
B	-
C	-
D	Large Regulated Motor

**Tabelle D.2:** Motorbelegung

---

## E Wichtige Funktionen

---

Hier eine kleine Übersicht über die wichtigsten Funktionen beim Programmieren der Roboter.

---

### E.1 Fahren

---

Mögliche Eingabewerte für den *speed*-Parameter liegen zwischen 0 und 1000. Eine maximale Geschwindigkeit von 300 sollte ausreichen. Niedrigere Geschwindigkeiten schonen den Akku. Die Distanz wird im *distance*-Parameter immer als Kommazahl in Zentimetern (cm) angegeben (z.B.: 20cm werden als 20.0f angegeben)

Typ	Methode und Beschreibung
void	setMotorSpeed(int speed)  Bestimmt die Geschwindigkeit für Fahrmethoden ohne <i>speed</i> -Parameter.
void void	forward() backward()  Fahren mit der von <i>setMotorSpeed(...)</i> gesetzten Geschwindigkeit.
void void	driveDistanceForward(float distance) driveDistanceBackward(float distance)  Fahren mit der von <i>setMotorSpeed(...)</i> gesetzten Geschwindigkeit Die Distanz muss in Zentimetern angegeben werden.
void void void void	forward(int speed) backward(int speed) driveDistanceForward(float distance, int speed) driveDistanceBackward(float distance, int speed)  Wie oben, nur dass der <i>speed</i> -Parameter die von <i>setMotorSpeed()</i> gesetzte Geschwindigkeit überschreibt. Nach Beendigung des Aufrufs, wird wieder die vorher gesetzte Geschwindigkeit genutzt.
void void void void	turnLeft(int degrees) turnRight(int degrees) turnLeft(int degrees, int speed) turnRight(int degrees, int speed)  Dreht den Roboter um den im <i>degrees</i> -Parameter bestimmten Wert. Der <i>Speed</i> -Parameter verhält sich wie bei den anderen Methoden.
void	stop()  Stoppt sofort alle Motoren.

## E.2 Sensoren

Typ	Methode und Beschreibung
float	<code>getAngle()</code>  Liefert den Winkel des Gyrosensors in Grad
float	<code>getDistance()</code>  Liefert die vom Ultraschallsensor gemessene Distanz in Zentimetern
Colors Colors	<code>getLeftColor()</code> <code>getRightColor()</code>  Liefert den Wert des Linken/Rechten Farbsensors Farbwerte: Colors.BLACK, Colors.BLUE, Colors.BROWN, Colors.GREEN, Colors.RED, Colors.WHITE, Colors.YELLOW, Colors.NONE

## E.3 Kommunikation

Typ	Methode und Beschreibung
boolean	<code>hasMessage()</code> Prüft ob Nachricht vorhanden ist
MindroidMessage	<code>getNextMessage()</code> Ruft nächste Nachricht ab
void	<code>broadcastMessage(String message)</code> Sendet eine Nachricht an alle Roboter
String	<code>getRobotID()</code> Gibt den Namen des Roboters zurück.
void	<code>sendLogMessage(String logmessage)</code> Sendet eine Nachricht an den Message Server
void	<code>sendMessage(String destination, String message)</code> Sendet eine Nachricht an den <i>destination</i> -Roboter

Um eine Nachricht zu empfangen, muss zuerst mit `hasMessage()` überprüft werden ob eine Nachricht vorhanden ist. Liefert `hasMessage()` true zurück, kann mit `getNextMessage()` eine Nachricht abgerufen werden. Das Beispiel in Listing 1 zeigt wie das geht.

```
1      if (hasMessage()) {  
2          String msg = getNextMessage().getContent();  
3      }
```

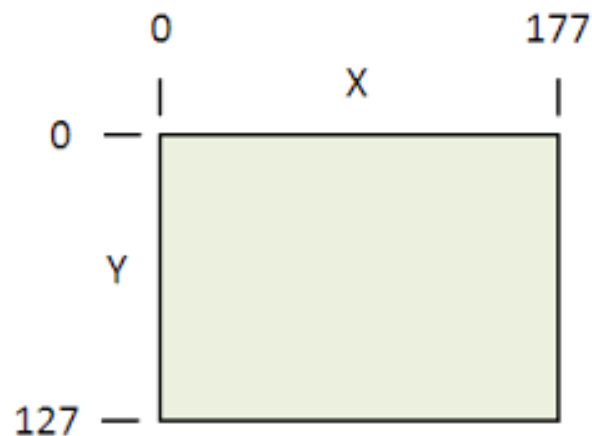
**Listing 1:** Beispiel zum Abrufen einer Nachricht

`broadcastMessage(...)` schickt eine Nachricht an alle mit dem selben Message-Server verbundenen Roboter.

## E.4 Brick

### E.4.1 Display

Typ	Methode und Beschreibung
void	<code>clearDisplay()</code>  Löscht den Aktuellen Inhalt des Displays
void	<code>drawString(String text, Textsize textsize, int xPosition, int yPosition)</code>  Schreibt den im <i>text</i> -Parameter gegebenen Text auf das Display an die durch <i>xPosition</i> und <i>yPosition</i> definierte Stelle (siehe Abb. E.1) mit Textgröße <i>textsize</i>



**Abbildung E.1:** Koordinaten der Pixel des Displays des EV3<sup>6</sup>

### E.4.2 Buttons

Typ	Methode und Beschreibung
boolean	<code>isDownButtonClicked()</code>
boolean	<code>isEnterButtonClicked()</code>
boolean	<code>isLeftButtonClicked()</code>
boolean	<code>isRightButtonClicked()</code>
boolean	<code>isUpButtonClicked()</code>

Die Funktionen liefern *true* wenn der entsprechende Button gedrückt wurde. Die Benennung der Buttons kannst du Abbildung A.1 auf Seite 18 entnehmen

<sup>6</sup> [https://services.informatik.hs-mannheim.de/~ihme/lectures/LEGO\\\_Files/01\\\_Anfaenger\\\_Graphisch\\\_EV3\\\_BadenBaden.pdf](https://services.informatik.hs-mannheim.de/~ihme/lectures/LEGO\_Files/01\_Anfaenger\_Graphisch\_EV3\_BadenBaden.pdf)

### E.4.3 Sound

Typ	Methode und Beschreibung
void	setSoundVolume(int volume)
void	playBeepSequenceDown()
void	playBeepSequenceUp()
void	playBuzzSound()
void	playDoubleBeep()
void	playSingleBeep()

Der Parameter *volume* nimmt Werte von 0 bis 10 entgegen.

### E.4.4 LED

Typ	Methode und Beschreibung
void	setLED(int mode)
	Lässt die LED des EV3 im angegebenen Modus leuchten Der Parameter <i>mode</i> kann entweder als Ganzzahl von 0 bis 9 oder als Konstante angegeben werden. Siehe Tabelle E.1

**Tabelle E.1:** Funktion der einzelnen Modi der LED

Wert	Modus (Parameter <i>mode</i> )	Farbe	Intervall
	Konstante		
0	LED_OFF	Aus	Aus
1	LED_GREEN_ON	Grün	Dauer
2	LED_GREEN_BLINKING	Grün	Blinken
3	LED_GREEN_FAST_BLINKING	Grün	Schnell Blinken
4	LED_YELLOW_ON	Gelb	Dauer
5	LED_YELLOW_BLINKING	Gelb	Blinken
6	LED_YELLOW_FAST_BLINKING	Gelb	Schnell Blinken
7	LED_RED_ON	Rot	Dauer
8	LED_RED_BLINKING	Rot	Blinken
9	LED_RED_FAST_BLINKING	Rot	Schnell Blinken