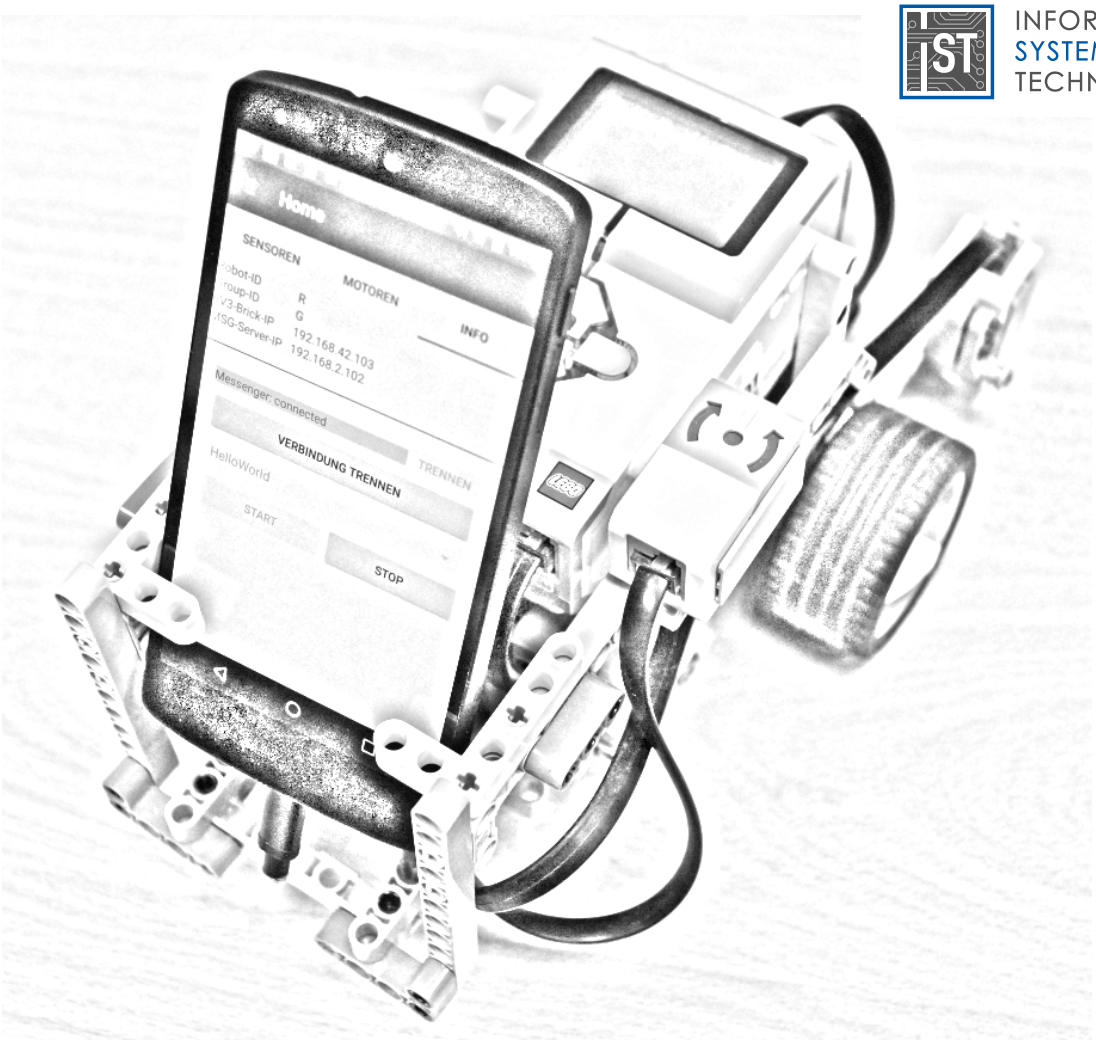
Mindroid Workshop Dokumentation

NeXT Generation on Campus
TU Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

INFORMATIONS SYSTEM TECHNIK



1 Einführung

In dieser Übersicht werden die Funktionen, die zur Steuerung der Roboter zur Verfügung stehen erklärt. Zur Verdeutlichung ein kleines Beispiel:

Typ	Methode und Beschreibung
void	delay(long milliseconds) Verzögert die Ausführung um die angegebene Zeitspanne (Milisekunden)

Die Spalte *Typ* gibt an, welchen Typ der Rückgabewert der Funktion hat. *void* bedeutet, dass kein Wert zurück gegeben wird. In der Klammer hinter dem Funktionsnamen wird angegeben, welche Parameter die Funktion erwartet, und von welchem Typ diese sein müssen. In unserem Beispiel bedeutet dies, dass die *delay*-Methode einen Parameter vom Typ *long* (ganzzahliger Wert) erwartet, welcher *milliseconds* genannt wird. Ein möglicher Funktionsaufruf sieht wie folgt aus:

```
1    public void run(){
2        delay(1000);
3    }
```

Dabei wird die delay-Methode mit 1000 als Parameter aufgerufen. Das bedeutet, die Ausführung wird um $1000ms$ ($= 1s$) verzögert.

1.1 isInterrputed

Damit die Ausführung des Programms auch in Schleifen unterbrochen werden kann, sollte jede Schleife die isInterrupted-Methode abfragen.

Beispiel:

```
1        public void run(){
2            while(!isInterrupted()){
3                // Schleifeninhalt
4            }
5            for(int i=0; i<10 && !isInterrupted(); i++){
6                // Schleifeninhalt
7            }
8        }
```

2 Wichtige Funktionen

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

2.1 Fahren

```
import org.mindroid.api.ImperativeWorkshopAPI
```

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 $speed$-Parameter.
void void	forward() backward() Fahren mit der von $setMotorSpeed(...)$ gesetzten Geschwindigkeit.
void void	driveDistanceForward(float distance) driveDistanceBackward(float distance) Fahren mit der von $setMotorSpeed(...)$ 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 $speed$-Parameter die von $setMotorSpeed()$ 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 $degrees$-Parameter bestimmten Wert. Der $Speed$-Parameter verhält sich wie bei den anderen Methoden.
void	stop() Stoppt sofort alle Motoren.

2.2 Sensoren

```
import org.mindroid.api.ImperativeWorkshopAPI
```

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

2.3 Kommunikation

```
import org.mindroid.api.ImperativeWorkshopAPI
```

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

2.4 MindroidMessage

Um die von $getNextMessage()$ zurückgegebene Nachricht verarbeiten zu können, muss ein zusätzlicher import hinzugefügt werden.

```
import org.mindroid.common.messages.server.MindroidMessage;
```

Typ	Methode und Beschreibung
String	getContent() Liefert den Inhalt der Nachricht zurück
RobotID RobotID	getDestination() getSource() Liefert die Quelle/das Ziel der Nachricht an

2.5 Brick

2.5.1 Display

Typ	Methode und Beschreibung
void	clearDisplay() Löscht den Aktuellen Inhalt des Displays
void void	drawString(String text) drawString(String text, int row) Schreibt den im $text$-Parameter gegebenen Text auf das Display. Der Parameter row bestimmt die zu beschreibende Zeile. Wird der Parameter row weggelassen, wir in die Mittlere Zeile geschrieben.

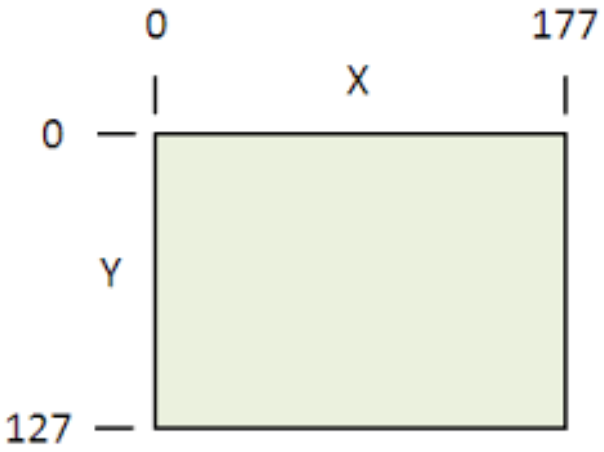


Abbildung 2.1: Koordinaten der Pixel des Displays des EV3[1]

[1] https://services.informatik.hs-mannheim.de/~ihme/lectures/LEGO_Files/01_Anfaenger_Graphisch_EV3_BadenBaden.pdf

2.5.2 Buttons

import org.mindroid.impl.brick.Button;

Typ	Methode und Beschreibung
boolean	isDownButtonClicked()
boolean	isEnterButtonClicked()
boolean	isLeftButtonClicked()
boolean	isRightButtonClicked()
boolean	isUpButtonClicked()

Die Funktionen liefern $true$ wenn der entsprechende Button gedrückt wurde. Die Benennung der Buttons kannst du Abbildung 3.1 auf Seite 8 entnehmen

2.5.3 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.

2.5.4 LED

Typ	Methode und Beschreibung
void	setLED(int mode) Lässt die LED des EV3 im angegebenen Modus leuchten Der Parameter $mode$ kann entweder als Ganzzahl von 0 bis 9 oder als Konstante angegeben werden. Siehe Tabelle 2.1

Tabelle 2.1: Funktion der einzelnen Modi der LED

Modus (Parameter $mode$)		**Farbe**	**Intervall**
Wert	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

3 EV3 Tasten

Abbildung 3.1 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.

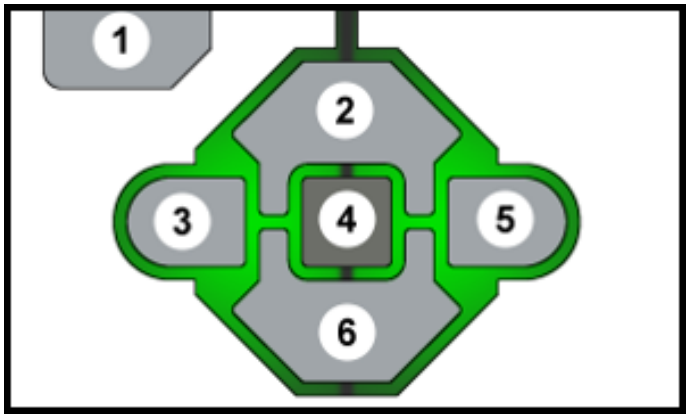


Abbildung 3.1: EV3-Tastenbelegung[2]

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**

4 Kurze Übersicht über Java

Schleife mit Bedingung	while(Bedingung) {Programmcode} Beispiel: while(i<100){...}
Zählschleife	for(Start; Bedingung; Zählschritte) {Programmcode} Beispiel: for(int i=0;i<10;i++){...}

Bedingung	if(Bedingung)
	{wenn die Bedingung wahr ist, wird dieser Code ausgeführt}
	else
	{wenn die Bedingung falsch ist, wird dieser Code ausgeführt}

5 Hello World

In der Informatik ist es üblich, ein Hallo-Welt-Programm[3] zu schreiben, wenn man eine Programmierumgebung kennenlernt. Deshalb fangen wir damit an. Nachdem du den Roboter erfolgreich eingerichtet und das erste Mal getestet hast, gehen wir jetzt daran, uns den Quelltext näher anzusehen.

[2] Quelle http://www.ev3dev.org/images/ev3/labeled-buttons.png
[3] https://de.wikipedia.org/wiki/Hallo-Welt-Programm

```java
import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.impl.brick.Textsize;

public class HelloWorld extends ImperativeWorkshopAPI {

    public HelloWorld() {
        super("Hello World [sol]");
    }

    @Override
    public void run() {
           clearDisplay();
           drawString("Hello World", Textsize.MEDIUM, 10 , 50);
    }
}
```

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.**

5.1 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.

```java
import org.mindroid.api.ImperativeWorkshopAPI;
import org.mindroid.api.ev3.EV3StatusLightColor;
import org.mindroid.api.ev3.EV3StatusLightInterval;
import org.mindroid.impl.brick.Textsize;

public class ParkingSensor extends ImperativeWorkshopAPI {

    public ParkingSensor() {
        super("Parking Sensor [sol]");
    }

  @Override
    public void run() {
        String previousState = "";
        clearDisplay();
        drawString("Parking sensor", Textsize.MEDIUM, 10, 10);
        while (!isInterrupted()) {
            clearDisplay();
            if(getDistance() < 30f && getDistance() > 15f) {
                drawString("Hm :-/", Textsize.MEDIUM, 10, 10);
                if (!previousState.equals("hm")) {
                    setLED(LED_YELLOW_BLINKING);
                }
                previousState = "hm";
            } else if (getDistance() < 15f) {
                drawString("Oh oh :-O", Textsize.MEDIUM, 10, 10);
                if (!previousState.equals("oh")) {
                    setLED(LED_RED_BLINKING);
                }
                previousState = "oh";
```

```
31              } else {
32                  drawString("OK :-)", Textsize.MEDIUM, 10, 10);
33                  if (!previousState.equals("ok")) {
34                      setLED(LED_GREEN_ON);
35                  }
36                  previousState = "ok";
37              }
38              delay(100);
39          }
40      }
41  }
```

- 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.

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

- 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?

5.3 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 [sol]");
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 [sol]");
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, 60);
18                 }
19             }
20             delay(100);
21         };
22     }
23   }
```

- Robert überprüft mit **`hasMessage()`** (Zeilte 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[4]. Stimmen beide überein, schreibt Robert auf sein Display einen Text (Zeile 19).

[4] Beachte: Strings werden in Java nicht mit == verglichen, sondern mittels der **`equals()`**-Methode