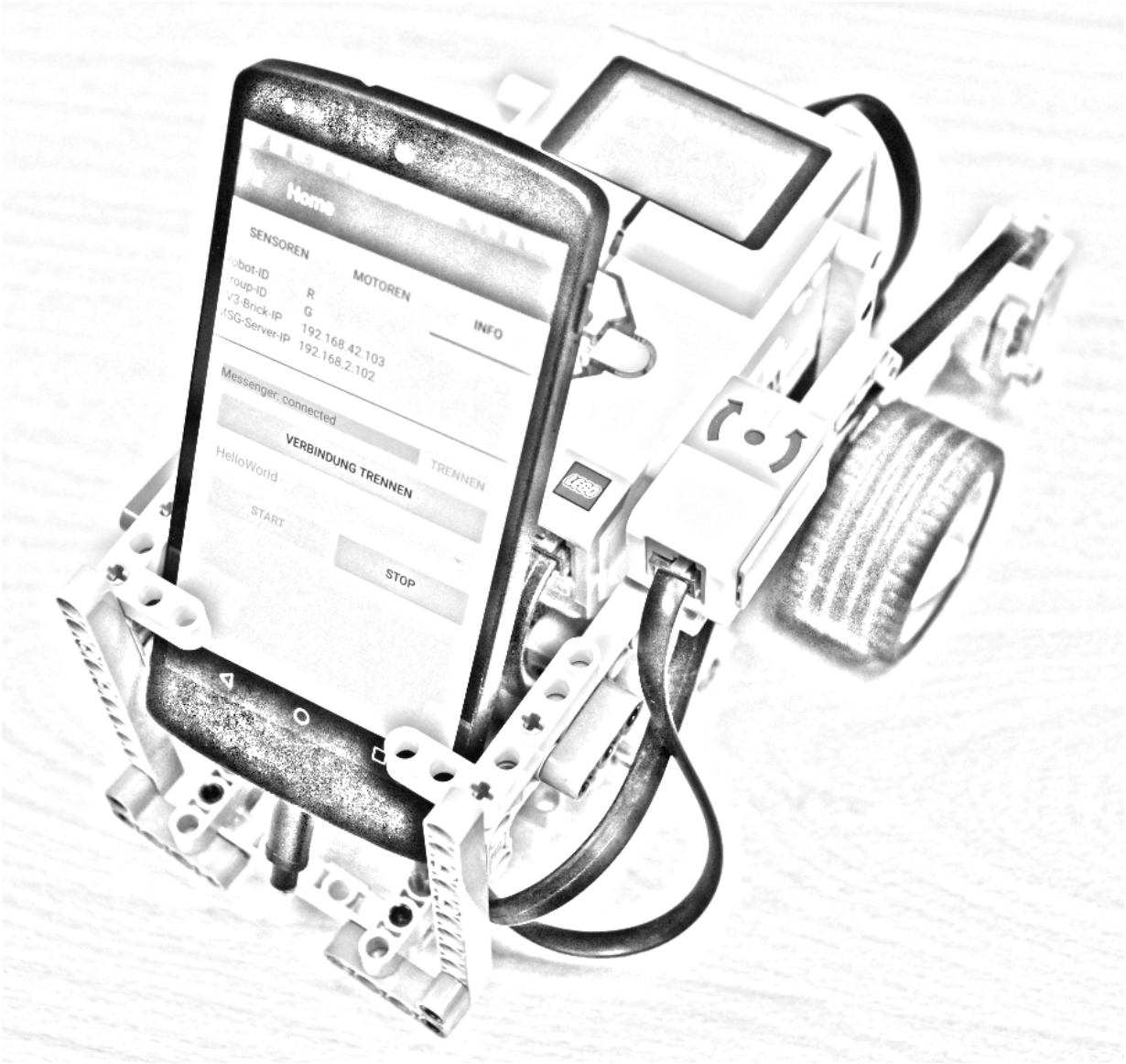
Mindroid Workshop Aufgabenstellung

TECHNISCHE UNIVERSITÄT DARMSTADT

NeXT Generation on Campus
TU Darmstadt



Aufgabe 1 Hello World

In der Informatik ist es üblich, ein Hallo-Welt-Programm[1] 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.

```
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 [sol]");
8      }
9
10     @Override
11     public void run() {
12             clearDisplay();
13             drawString("Hello World");
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 Mindroid-App 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 1.1 HelloDate

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] https://de.wikipedia.org/wiki/Hallo-Welt-Programm

```
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 [sol]");
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()));
17             }
18   }
```

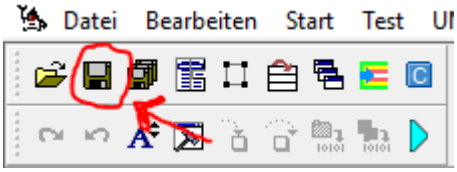
- Die Imports von **java.text.SimpleDateFormat** und **java.util.Date** sind neu hinzugekommen.
- Außerdem gibt es nun eine Variable **formatter** (Zeile 17), die in der Lage ist, das aktuelle Datum formatiert auszugeben (**formatter.format(new Date())**, Zeile 19)).

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

1. Zuerst musst du deine Änderungen am Code speichern[2].
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”

[2] Oben Links auf das Diskettensymbol klicken 

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

Aufgabe 1.2 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 [sol]");
10     }
11
12   @Override
13     public void run() {
14         String previousState = "";
15         clearDisplay();
16         drawString("Parking sensor");
17         while (!isInterrupted()) {
18             clearDisplay();
19             if(getDistance() < 30f && getDistance() > 15f) {
20                 drawString("Hm :-/");
21                 if (!previousState.equals("hm")) {
22                     setLED(LED_YELLOW_BLINKING);
23                 }
24                 previousState = "hm";
25             } else if (getDistance() < 15f) {
26                 drawString("Oh oh :-O");
27                 if (!previousState.equals("oh")) {
28                     setLED(LED_RED_BLINKING);
29                 }
30                 previousState = "oh";
31             } else {
32                 drawString("OK :-)");
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.

Aufgabe 1.3 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 [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", 3);
19             drawString("L: " + describeColor(leftColorValue), 4);
20             drawString("R: " + describeColor(rightColorValue), 5);
21             drawString("Distance: " + getDistance(), 6);
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?

Aufgabe 1.4 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  import org.mindroid.impl.brick.Textsize;
4
5  public class HelloWorldPingA extends ImperativeWorkshopAPI {
6
7      public HelloWorldPingA() {
8          super("Hello World Ping Alice [sol]", 2);
```

```
9       }
10
11      @Override
12      public void run() {
13          clearDisplay();
14          while(!isInterrupted()){
15              delay(10);
16              if(isButtonClicked(Button.ENTER)) {
17                  sendMessage("Bob", "Hallo Bob!");
18                  drawString("Nachricht an Bob gesendet!");
19                  sendLogMessage("Nachricht an Bob gesendet!");
20              }
21          }
22      }
23  }
```

- 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 Bob [sol]", 2);
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 Bob!")){
17                      drawString("Nachricht von Alice erhalten");
18                      sendLogMessage("Nachricht von Alice erhalten!");
19                  }
20              }
21              delay(100);
22          };
23      }
24    }
```

- 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[3]. Stimmen beide überein, schreibt Robert auf sein Display einen Text (Zeile 19).

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

Aufgabe 2 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 heranfahren, ohne mit ihr zu kollidieren.
2. Dann soll er ein kleines Stück rückwärts fahren und sich um 180° drehen
3. Beginne bei 1.

Aufgabe 3 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.

Aufgabe 4 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.

Aufgabe 5 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 Vorausfahrenden 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 bestimmen 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).

Aufgabe 6 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°.
5. Nun tauschen Roboter A und B die Rollen: Roboter B fährt voraus und gibt den Ton bis zur nächsten Drehung an.