

- [Einführung-Begrüßung](#)
- [Einführung-Übersicht](#)
- [Einführung-Benutzeroberfläche](#)
- [Einführung-Navigation](#)
- [Einführung-Lektionsaufbau](#)
- [Einführung-Theory Task](#)
- [Einführung-MultipleChoiceTask](#)
- [Einführung-BugMultipleChoiceTask](#)
- [Einführung-EducationalTask](#)
- [Einführung-Hint](#)
- [Einführung-RunConfiguration](#)
- [Setup-Einführung](#)
- [Setup-Ausführung](#)
- [Erste Komponente-Übersicht](#)
- [Erste Komponente-Selbsteinschätzung](#)
- [Erste Komponente-Hardwarekatalog](#)
- [Erste Komponente-Pins Raspberry Pi](#)
- [Erste Komponente-Eventhandler](#)
- [Erste Komponente-Implementation Button](#)
- [Erste Komponente-Implementation LED](#)
- [Erste Komponente-Zusammenfassung](#)
- [Erste Komponente-Validierung](#)
- [Klassenmodifizierung-Übersicht](#)
- [Klassenmodifizierung-Selbsteinschätzung](#)
- [Klassenmodifizierung-Klassenmodifizierung](#)
- [Klassenmodifizierung-Implementation Klassenmodifizierung](#)
- [Klassenmodifizierung-Wiederverwendbarkeit](#)
- [Klassenmodifizierung-Implementation Klassenwiederverwendung](#)
- [Klassenmodifizierung-Zusammenfassung](#)
- [Klassenmodifizierung-Validierung](#)
- [PI4J-Umgebung-Übersicht](#)
- [PI4J-Umgebung-Selbsteinschätzung](#)
- [PI4J-Umgebung-Einleitung](#)
- [PI4J-Umgebung-Standardcontext](#)
- [PI4J-Umgebung-Provider](#)
- [PI4J-Umgebung-Providerwahl](#)
- [PI4J-Umgebung-Shutdown](#)
- [PI4J-Umgebung-Shutdownimplementation](#)
- [PI4J-Umgebung-Zusammenfassung](#)
- [PI4J-Umgebung-Validierung](#)
- [ModelViewController-Übersicht](#)
- [ModelViewController-Selbsteinschätzung](#)
- [ModelViewController-ModelViewController](#)
- [ModelViewController-View](#)
- [ModelViewController-Kontroller](#)
- [ModelViewController-Model](#)
- [ModelViewController-Zusammenfassung](#)
- [ModelViewController-Validierung](#)
- [Eigenes Projekt-Übersicht](#)
- [Eigenes Projekt-Einleitung](#)
- [Eigenes Projekt-Projektstart](#)
- [Eigenes Projekt-Schema](#)
- [Eigenes Projekt-SimonSays](#)
- [Eigenes Projekt-Zusammenfassung](#)
- [Eigenes Projekt-Validierung](#)

## Leitfaden zu Pi4J Anwendungen



Der vorliegende Leitfaden soll in die Themen von PI4J, dem Hardware-Katalog, dem MVC-Pattern sowie dem Template-Projekt für PI4J einführen. Klick auf den Button *Next* um mit dem Tutorial zu starten.

## Task 1/10: Übersicht

In dieser Lektion sollen die Funktion und die Bedienung des *Leitfadens* erklärt werden. Dazu sind folgende Themen beschrieben:

### Lernziele

Wenn alle Fragen mit Ja beantwortet werden können, kann das Kapitel übersprungen werden, und mit der nächsten Lektion weitergemacht werden.

- Aufbau der Benutzeroberfläche
- Navigation im Leitfaden
- Aufbau einer Lektion
- Erklärung eines *Theory Task*
- Erklärung eines *Multiple Choice Task*
- Erklärung eines *Educational Task*
- Was ist ein *Hint*
- Was ist eine *Run Configuration*

[Sprung zur nächsten Lektion: Setup](#)

### Zeitschätzung

Diese Lektion wird etwa 20 Minuten in Anspruch nehmen.

Mit dem Button *Next* wird die Lektion *Einführung* gestartet.

## Task 2/10: Benutzeroberfläche

Die Grundeinstellung des Leitfadens sieht wie folgt aus:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // Write your solution here  
4     }  
5 }
```

1

Project Description Structure Bookmarks

Task 2 Benut

Die Grundei

1. Projekt  
2. Editor  
3. Aufgabe  
4. Navigation  
5. Smart Checker

Run Done

1. Editor
2. Aufgabenfenster
3. Navigationspfeile
4. Smart Checker

## Editor

Der Leitfaden beinhaltet diverse praktische Programmieraufgaben, welche im Editor direkt gelöst werden können.

## Aufgabenfenster

Im Aufgabenfenster wird, je nach Aufgabe, die Theorie zur Lektion vermittelt, Multiple-Choice-Fragen zur Lektion oder praktische Aufgaben gestellt, welche im Editor gelöst werden müssen.

## Navigationspfeile

Mit den Navigationspfeilen kann zwischen den einzelnen Tasks einer Lektion hin- und hergesprungen werden.

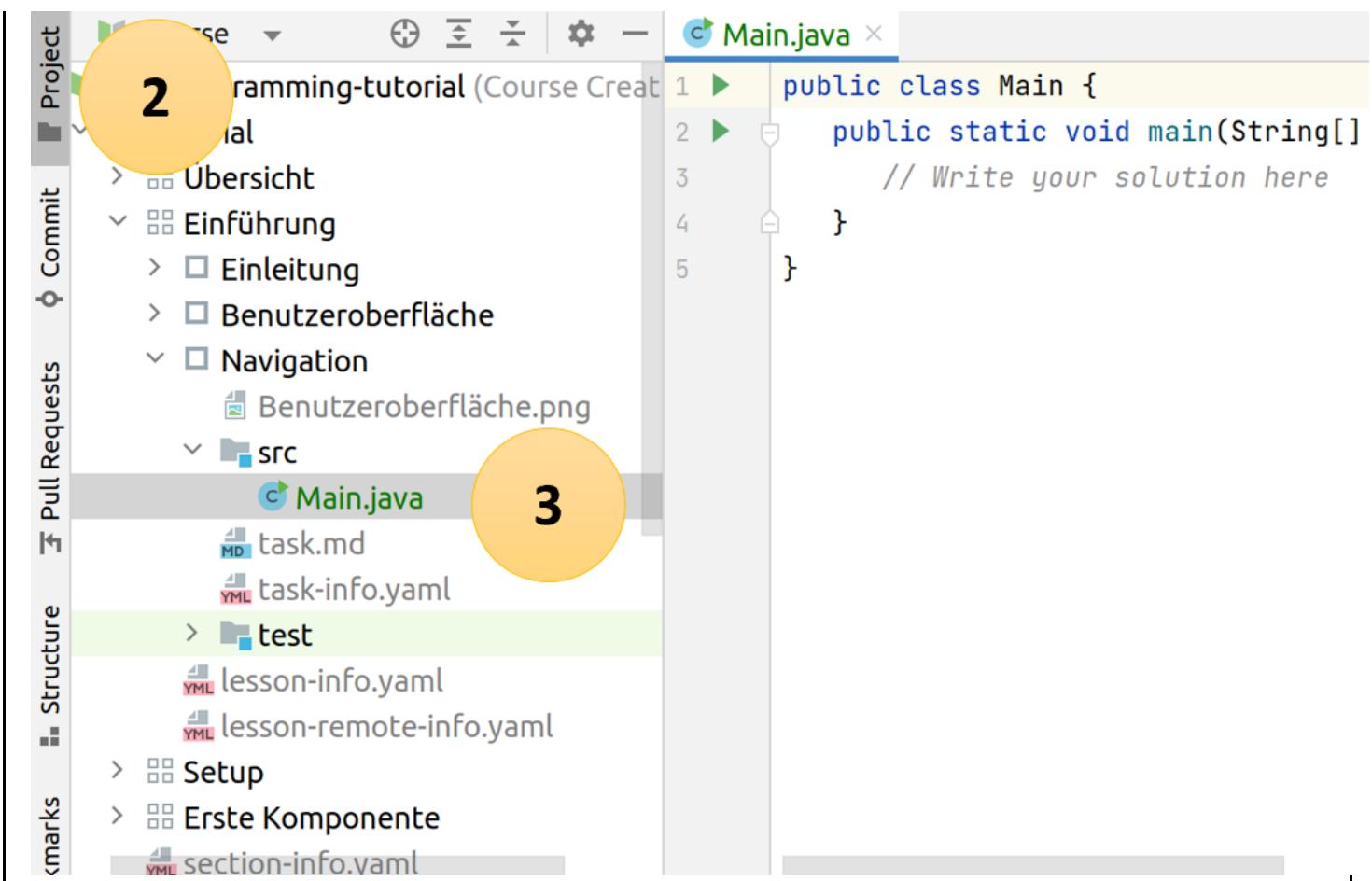
## Smart Checker

Mit dem Smart Checker kann die Lösung überprüft und zum nächsten Task in der Lektion gesprungen werden.

Mit dem Button *Next* zum nächsten Task wechseln.

## Task 3/10: Navigation

Mit den Pfeiltasten (1) kann zum nächsten Task bez. zum vorangehenden Task gewechselt werden, auch wenn der aktuelle Task nicht gelöst ist.



Mit dem Reiter *Project* (2) kann der Projektbaum geöffnet werden. Im Projektbaum sind die einzelnen Lektionen des Leitfadens aufgelistet. Alternativ zu den *Next* Button kann eine Lektion auch über den Projektbaum gestartet werden. Dazu einfach die Klasse *Main.java* (3) des entsprechenden Tasks im Editor per Doppelklick öffnen.

Mit dem Button *Next* zum nächsten Task wechseln.

## Task 4/10: Lektionsaufbau

Der Leitfaden ist in die unten aufgelisteten Lektionen unterteilt:

1. Einführung
2. Setup
3. Erste Komponente
4. Klassenmodifizierung
5. Einführung in die Pi4J Umgebung
6. Einführung in das MVC-Pattern
7. Erstellen einer eigenen Applikation

Jede Lektion beinhaltet die folgenden Tasks:

### Übersicht

Jede Lektion beginnt mit einer Übersicht über die Inhalte, welche in dieser Lektion behandelt werden und wie viel Zeit das Lösen der Lektion in etwa in Anspruch nimmt.

### Selbsteinschätzung

Die Selbsteinschätzung ist eine Sammlung von Kontrollfragen über die Inhalte der Lektion. Können alle Aufgaben richtig beantwortet werden, sind die Inhalte der Lektion bereits bekannt und die Lektion kann übersprungen werden.

### Einleitung in die Thematik

Die Einleitung ist der Theorieteil der Lektion. Hier wird das nötige Wissen vermittelt um die folgenden praktischen Aufgaben zu lösen.

### Praktische Aufgaben

Die praktischen Aufgaben stellen den Übungsteil der Lektion dar. Hier wird der Stoff aus der Einleitung vertieft.

## Zusammenfassung

In der Zusammenfassung werden die wichtigsten Informationen aus der Lektion noch einmal kurz zusammengefasst.

## Validierung

Die Validierung dient als Überprüfung, ob die Themen der Lektion verstanden wurden. Können alle Fragen richtig beantwortet werden, kann mit der nächsten Lektion begonnen werden. Bestehen bei einzelnen Fragen noch Unklarheiten, können diese mit den Betreuungspersonen geklärt werden, oder der entsprechende Task kann erneut durchgearbeitet werden.

Mit dem Button *Next* zum nächsten Task wechseln.

## Task 5/10: Theory Task

Der Leitfaden ist in einzelne Lektionen unterteilt und jede Lektion hat unterschiedliche Tasks. Es gibt drei unterschiedliche Arten von Tasks. Den *Theory Task*, den *Multiple Choice Task* und den *Educational Task*.

Der **Theory Task** ist bereits aus den vorangehenden Tasks bekannt und wird verwendet um grundlegendes Wissen, wie zum Beispiel in der Einleitung, zu vermitteln.

Der Bereich *Smart Checker* hat zwei Buttons, *Run* und *Next*.

**Run** dient dazu, allfällige Codebeispiele welche im Editor programmiert sind, auszuführen. Mit dem *Run* Button kann nicht zur nächsten Lektion gewechselt werden. Dazu gibt es den Button *Next*.

**Next** dient dazu, zum nächsten Task zu wechseln.

## Task 6/10: MultipleChoiceTask

Im **Multiple-Choice Task** werden Fragen zur aktuellen Lektion gestellt. Die möglichen Antworten sind im Bereich des Smart Checkers aufgelistet. Wie im *Theory Task* gibt es auch hier einen Button *Next*, um zum nächsten Task zu wechseln. Zudem gibt es den Button **Check**, mit welchem die Antworten der Multiple-Choice-Frage gleich überprüft werden können. Um mit *Next* zum nächsten Task zu wechseln, muss zuerst der Check erfolgreich sein.

Beantworte die folgende Frage richtig, um zum nächsten Task wechseln zu können:

Was ist der Sinn des Lebens?

Suche mit Google nach "Sinn des Lebens"

## Task 7/10: MultipleChoiceTask

Bei Multiple-Choice-Fragen mit viereckigen Checkboxen (siehe unten) sind mehrere Antworten richtig. Die Aussage *Select one option from the list* kann leider nicht geändert werden. Es gilt:

- runde Checkbox -> nur eine Option ist richtig
- viereckige Checkbox -> mehrere Optionen können richtig sein

## Task 8/10: EducationalTask

Der **Educational Task** ist ein Task, um Programmieraufgaben zu lösen. Jeder Task hat ein Main.java File mit unterschiedlichen Aufgaben. Mit Platzhaltern ist ersichtlich, welche Aufgabe an welcher Stelle im Code gelöst werden muss. Im Aufgabenfenster sind alle wichtigen Informationen zu den Aufgaben vorhanden.

Im *Smart Checker* gibt es neben dem bekannten *Next* Button einen Button **Check**. Mit diesem Button lässt sich der erstellte Code überprüfen.

Nachdem ein Check durchgeführt wurde, kann mit Peak Solution (1) die Musterlösung betrachtet werden.

[Check](#)[Next](#)

✓ Correct Moments ago

Congratulations!

[Peek Solution...](#)

1

## Aufgabe

Gib den Text *Hello world, hello Leitfaden* auf der Konsole aus.

Benutze dazu den String `*text*` und `*System.out.println()*`.

Überprüfe mit *Check* den Code. Wenn dieser richtig funktioniert, kann mit *Next* zur nächsten Lektion navigiert werden.

## Task 9/10: Hint

An diversen Stellen im Leitfaden sind Hilfestellungen, sogenannte *Hints* platziert. Diese sind eine Unterstützung, die richtige Lösung zu finden. Benutze den untenstehenden Hint, um die Multiple-Choice-Frage richtig zu beantworten. Hints können per Klick geöffnet werden.

Der Begriff "Computer Bug" wurde in der Tat von einem echten Insekt inspiriert.

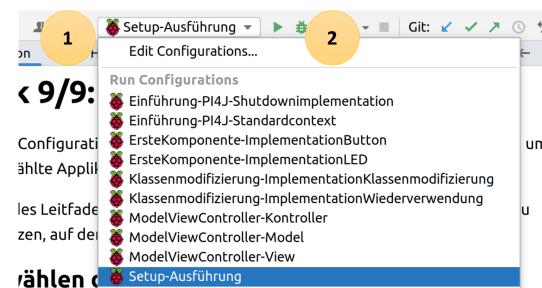
## Task 10/10: RunConfiguration

Die RunConfiguration ist eine Zusammenstellung von verschiedenen Befehlen, um eine ausgewählte Applikation auszuführen.

Im Fall des Leitfadens wird die RunConfiguration benutzt, um die Applikation zu übersetzen, auf dem Raspberry Pi bereitzustellen und auszuführen.

### Auswählen der RunConfiguration

Um die Applikation auf dem Raspberry Pi auszuführen, muss die entsprechende RunConfiguration im Drop Down Menü (1) ausgewählt und danach mit dem grünen Pfeil (2) gestartet werden.

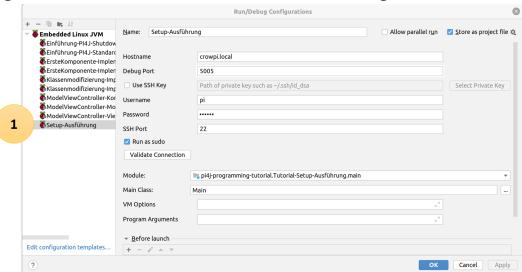


### Anpassen der RunConfiguration

Muss eine Konfiguration angepasst werden, weil zum Beispiel die IP-Adresse gewechselt werden muss, kann im Drop Down Menü *Edit Configuration* angewählt



RunConfiguration auswählen. Danach lassen sich die gewünschten Parameter editieren. Zum Beispiel kann anstelle des Hostname eine IP-Adresse vergeben



werden.

Es kann mit dem Button *Next* zur nächsten Lektion gewechselt werden.

## Task 1/2: Einführung

Diese Lektion dient der Überprüfung, ob alle Prerequisites erfüllt sind, ob die Hilfestellungen in Bezug auf die PI4J-Library bekannt sind, und ob das Tutorial richtig verwendet werden kann.

### Lernziele

- Ich kann per Remote-Ausführung meinen Code auf dem Raspberry PI testen.

### Zeitschätzung

Diese Lektion wird etwa 20 Minuten in Anspruch nehmen.

### Aufbau

Der Aufbau dieser Lektion sieht wie folgt aus:

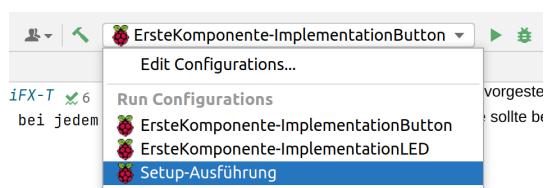
- Übersicht - *Theorie*
- Ausführung - *Educational*

## Task 2/2: Ausführung

In diesem Task wird eine kleine Applikation mit PI4J aufgezeigt, welche auf dem Raspberry PI ausgeführt werden soll. Was die Applikation genau macht, wird in einer späteren Lektion genauer erklärt.

### Remote Ausführung

In diesem Task wurde eine Running-Config hinterlegt. Diese kann wie hier im Bild aufgezeigt ausgewählt werden:



Mit der richtigen ausgewählten Running-Config, kann das Programm gestartet werden. Die Applikation wird auf das Raspberry Pi geladen und automatisch ausgeführt. Folgende bekannte Probleme können auftreten:

- Benutzername und Passwort des Raspberry Pi stimmen nicht mit der Running-Config überein -> PW auf Pi oder in Running-Config anpassen.
- Im gleichen Netzwerk gibt es mehr als ein Pi mit dem Namen Crowpi.local oder keinen -> IP-Adresse anstelle von Namen in der Running-Config verwenden.

## Aufgabe

### Programmierung

Im Code auf der linken Seite wird zuerst der Pi4J Context erstellt. Danach wird auf der Konsole die beiden Strings: "Application is running" und "Application is done" ausgegeben, bevor das Programm wieder endet.

**Hinweis:** Der Code wird zu einem späteren Zeitpunkt genau erklärt und muss hier nicht verstanden werden.

## Check Programmierung

Überprüfe die Programmierung mit dem *Check* Button unten links.

## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config *Setup-Ausführung* hinterlegt.

Konnte die Applikation erfolgreich auf dem Raspberry Pi ausgeführt werden kann mit dem Button *Next* zum nächsten Task gewechselt werden.

# Task 1/9: Übersicht

Diese Lektion gibt eine Einführung in den Hardwarekatalog und wie dieser in einer Applikation eingesetzt werden kann.

## Lernziele

- Ich kenne den groben Inhalt des Hardwarekataloges und weiss, wie ich diesen in meinem Projekt einsetzen kann.
- Ich weiss, wo ich im Hardwarekatalog Hilfe zur elektrischen Montage der Bauteile bekomme.
- Ich kenne die Funktionen von SimpleLed und SimpleButton und kann diese in einer einfachen Applikation verwenden.
- Ich kann eine einfache Applikation auf dem Raspberry Pi starten.

## Zeitschätzung

Diese Lektion wird etwa 90 Minuten in Anspruch nehmen.

## Aufbau

Der Aufbau dieser Lektion sieht wie folgt aus:

- Übersicht - *Theorie*
- Selbsteinschätzung - *MultipleChoice*
- Hardwarekatalog - *Theorie*
- Pins Raspberry Pi - *Theorie*
- Eventhandler - *Theorie*
- Implementation Button - *Educational*
- Implementation LED - *Educational*
- Zusammenfassung - *Theorie*
- Validierung - *MultipleChoice*

# Task 2/9: Selbsteinschätzung

Die Selbsteinschätzung dient dazu zu klären, ob bereits ausreichend Wissen zu den Themen der Lektion vorhanden ist, damit diese übersprungen werden kann.

Wenn alle unten aufgelisteten Multiple-Choice-Fragen beim ersten Mal richtig beantwortet werden, kann diese Lektion mit dem Link übersprungen werden. Bei Unklarheiten sind die korrekten Lösungen als *Hint* hinterlegt.

Nein, der Hardware-Katalog besteht aus der Software-Klasse, einer Anwendung der Klasse und einem Beispiel der Verdrahtung.  
Ja, der Hardware-Katalog besteht auch aus einem Schema.

Nein, aufgrund der Event Handler in der Komponente sollte diese nicht mehr angepasst werden müssen.  
Ja, das PI liefert 3,3V Spannung, die rote LED als Beispiel braucht maximal 1,8V.

[Wenn alles verstanden wurde, kann diese Lektion übersprungen werden.](#), ansonsten kann die Lektion mit dem Button *Next* gestartet werden.

# Task 3/9: Hardwarekatalog

Der Hardwarekatalog stellt zu unterschiedlichen elektronischen Bauteilen wie zum Beispiel einem LCD-Display, einem LED-Streifen oder einem Servo-Motor diverse Hilfestellungen zur Verfügung. Er ist ein Teil von [The Pi4J Project](#) und lässt sich grob in zwei Teile unterteilen.

Der erste Teil ist das Repository [Java I/O Library for Raspberry Pi](#), welches zu jedem Bauteil eine passende [Java Klasse](#) zur Verfügung stellt. Zudem gibt es zu jeder Klasse ein [Beispiel](#) welches die Implementation der Klasse erklärt.

**Hinweis:** Benötigte Java Klassen können aus dem Katalog kopiert und im Programm an der richtigen Stelle eingefügt werden.

Der zweite Teil des Katalogs bildet die Dokumentation auf der [Pi4J](#) Webseite. Das Kapitel [Component Examples](#) ist eine Auflistung sämtlicher im Katalog umgesetzter Bauteile. Zu jedem Bauteil gibt es einen detaillierten Eintrag über die wichtigsten Funktionen der Java Klasse, ein elektrisches Layout und den Java-Code der Beispielapplikation.

Neben der Beschreibung der einzelnen Bauteile verlinkt die Webseite zusätzlich auf zwei grössere Applikationen, [Photobooth](#) und [Theremin](#), welche beide mit Bauteilen aus dem Hardwarekatalog realisiert wurden.

## Task 4/9: Pins Raspberry Pi

Dieser Task gibt eine kurze Einführung in die GPIO Pins des Raspberry Pi.

### Raspberry PI PINs

Der GPIO ist der grundlegendste, aber dennoch zugänglichste Aspekt des Raspberry Pi. GPIO-Pins sind digital, was bedeutet, dass sie zwei Zustände haben können, aus oder an. Sie können eine Richtung zum Empfangen oder Senden von Strom haben (Eingang bzw. Ausgang) und wir können den Zustand und die Richtung der Pins steuern. Die Betriebsspannung der GPIO-Pins beträgt 3,3 V bei einer maximalen Stromaufnahme von 16 mA. Dies bedeutet, dass wir eine oder zwei LEDs sicher von einem einzelnen GPIO-Pin über einen Widerstand mit Strom versorgen können. Aber für alles, was mehr Strom benötigt, zum Beispiel einen Gleichstrommotor, müssen wir externe Komponenten verwenden, um sicherzustellen, dass wir den GPIO nicht beschädigen.

Die Pin-Nummerierung von Broadcom (BCM) (auch bekannt als GPIO-Pin-Nummerierung) scheint für den nicht versierten Benutzer chaotisch zu sein. Das BCM-Pin-Mapping bezieht sich auf die GPIO-Pins, die direkt mit dem System on a Chip (SoC) des Raspberry Pi verbunden wurden. Im Wesentlichen haben wir direkte Verbindungen zum Gehirn unseres Pi, um Sensoren und Komponenten für die Verwendung in unseren Projekten zu verbinden.

Die meisten Raspberry Pi-Tutorials verwenden die BCM-Nummerierung, da diese das offiziell unterstützte Pin-Nummerierungsschema der Raspberry Pi Foundation ist. Es empfiehlt sich daher, mit der Verwendung und dem Erlernen des BCM-Pin-Nummerierungsschemas zu beginnen, da es Ihnen mit der Zeit in Fleisch und Blut übergehen wird. Beachten Sie auch, dass sich die Pin-Nummerierung von BCM und GPIO auf dasselbe Schema bezieht. So ist zum Beispiel GPIO17 dasselbe wie BCM17. [Weitere Informationen](#)

### Raspberry PI Ein- und Ausgänge

Bestimmte GPIO-Pins haben auch alternative Funktionen, die es ihnen ermöglichen, mit verschiedenen Arten von Geräten zu kommunizieren welche die I2C-, SPI- oder UART-Protokolle verwenden. Zum Beispiel sind GPIO3 und GPIO4 auch SDA- und SCL-I2C-Pins, die verwendet werden, um Geräte mit dem I2C-Protokoll zu verbinden. Hier die [Theorie](#) dazu.

#### Digital Output

Ein digitaler Output übersetzt ein Falsch / Wahr (oder 0 / 1) in einen Ausgangswert von 0V oder 3,3V. Das bedeutet, dass jede Art von Gerät, das mit maximal 3,3V arbeitet, ein- oder ausgeschalten werden kann. Das einfachste Beispiel ist eine LED. Es muss immer geprüft werden, welches die richtige Eingangsspannung für das Gerät ist!

#### Digital Input

Ähnlich wie ein Digital Output-PIN übersetzt ein digitaler Eingang einen Eingangswert von 0V oder 3,3V in den Wert Falsch / Wahr. Das bedeutet, dass jede Art von Gerät, das zwischen 3,3V und 0V umschalten kann, einen Eingangswert für den Raspberry Pi erzeugen kann. Hier ist das einfachste Beispiel ein Schalter oder Taster. Wenn andere Komponenten verwendet werden, muss immer geprüft werden, welche Spannung das Gerät liefert. Wenn ein Power-Pin vom Raspberry Pi selbst verwendet wird, ist es ratsam, einen 3,3V-Pin und anstatt einem 5V-Pin zu verwenden.

## Task 5/9: Einführung Eventhandler

Mit einem Event Handler kann ein Softwareentwickler genau steuern, was im Programm geschehen soll, wenn ein bestimmtes Ereignis eintritt. Die auslösenden Events können dabei unterschiedlichen Ursprungs sein, oft werden sie durch die Interaktion des Anwenders ausgelöst.

Im Hardwarekatalog werden ebenfalls Event Handler eingesetzt. So wird zum Beispiel bei *SimpleButton* ein Event ausgelöst, wenn der Taster gedrückt wird oder wenn er wieder losgelassen wird.

Welchen Event das Drücken des Tasters auslöst, ist von seiner Programmierung abhängig. Die Klasse *SimpleButton* bietet ein Handler für den Event *Button ist gedrückt* und für *Button ist wieder los gelassen*. Mit diesen Handler können dem Objekt applikationsspezifische Aufgaben übergeben werden, ohne den Code in *SimpleButton* selber anpassen zu müssen.

Im unten stehenden Code wird die Methode `userFunction` von der Klasse `User` ausgeführt, wenn der Taster gedrückt wird.

```
button.onDown(() -> user.userFunction());
```

**Hinweis:**

mit ()-> wird ein neuer Thread gestartet in welchem die Methode `user.userFunction()` läuft. Die genaue Funktion muss an dieser Stelle noch nicht verstanden werden. Es gilt einfach die Syntax:

```
onDown( () -> hier kann eine Funktion aufgerufen werden );
```

## Task 6/9: Implementation Button

In dieser Aufgabe geht es darum, das Signal eines Tasters (gedrückt) auszuwerten. Dazu wird die Klasse [SimpleButton](#) aus dem Hardwarekatalog verwendet. Auf das Ereignis `Button gedrückt` soll sich der Zustand der LED ändern.

**Hinweis:**

Die Klasse `SimpleButton` wurde bereits aus dem Hardwarekatalog kopiert und in den Ordner src eingefügt.

## Aufgabe

### Programmierung

- Deklariere das Objekt `button` vom Typ Simple Button
- Initialisiere das Objekt `button`. Verwende dazu den Pin 26 auf dem Raspberry Pi
- Wenn der Button gedrückt wird, soll sich der Zustand der LED ändern. Verwende dazu die Methode `toggleState` von `SimpleLed`
- Nachdem die Zeit von 15s abgelaufen ist (nach `sleep(15000)`) soll die LED ausgeschalten werden, damit zum Schluss ein definierter Zustand erreicht wird
- Als letzte Ergänzung soll der Handler wieder deregistriert werden.

**Hinweis:**

Die Aufgabe kann zwischen den Zeilen **Start coding space** und **End coding space** gelöst werden. Was oberhalb und unterhalb dieser Zeilen steht wird benötigt, damit die Applikation funktioniert. Die genaue Bedeutung wird zu einem späteren Zeitpunkt erklärt.

Eine Beispianwendung ist bei der Komponente [Simple Button](#) auf der Pi4J Webseite programmiert.

### Check Programmierung

Überprüfe die Programmierung mit dem **Check** Button unten links. Ist der Test erfolgreich, kannst du mit der Vorbereitung der Hardware beginnen. Ist der Test nicht erfolgreich, versuche den Fehler zu beheben. Nutze dazu allfällige Hints oder schau in der Lösung nach.

### Aufbau

Als Nächstes muss der Taster mit dem Raspberry Pi richtig verbunden werden. Nutze dazu die Dokumentation [SimpleButton](#) des Hardwarekataloges auf der Pi4J Webseite.

### Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config `ErsteKomponente-ImplementationButton` hinterlegt.

## Task 7/9: Implementation LED

In dieser Aufgabe geht es darum, eine LED anzusteuern. Dazu wird die Klasse [SimpleLed](#) aus dem Hardwarekatalog verwendet.

**Hinweis:**

Die Klasse `SimpleLed` wurde bereits aus dem Hardwarekatalog kopiert und in den Ordner src eingefügt.

# Aufgabe

## Programmierung

- Deklariere das Objekt `led` vom Typ `SimpleLed`
- Initialisiere das Objekt `led`. Verwende dazu den Pin 19 (PWM Pin) auf dem Raspberry Pi
- Schalte die LED ein
- Blinklicht
  - Schalte die LED fünfmal ein und aus
  - Nutze dazu die Methode `toggleState` von `SimpleLed` in einem `for-loop`
  - Nutze die Methode `sleep()` um jedes Mal 1s zu warten, bis die LED den Zustand wechselt
- Schalte die LED aus

### Hinweis:

Die Aufgabe kann zwischen den Zeilen **Start coding space** und **End coding space** gelöst werden. Was oberhalb und unterhalb dieser Zeilen steht wird benötigt, damit die Applikation funktioniert. Die genaue Bedeutung wird zu einem späteren Zeitpunkt erklärt.

Eine Beispianwendung ist bei der Komponente [SimpleLed](#) auf der Pi4J Webseite programmiert.

## Check Programmierung

Überprüfe die Programmierung mit dem *Check* Button unten links. Ist der Test erfolgreich, kannst du mit der Vorbereitung der Hardware beginnen. Ist der Test nicht erfolgreich, versuche den Fehler zu beheben. Nutze dazu allfällige Hints oder schau in der Lösung nach.

## Aufbau

Als Nächstes muss die LED mit dem Raspberry Pi richtig verbunden werden. Nutze dazu die Dokumentation [SimpleLed](#) des Hardwarekataloges auf der Pi4J Webseite.

## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config *ErsteKomponente-ImplementationLED* hinterlegt.

# Task 8/9: Zusammenfassung

Dieser Tasks beinhaltet eine kurze Zusammenfassung, was in dieser Lektion alles abgedeckt wurde.

## Hardwarekatalog

Für die Unterstützung in den Projekten IP12 steht den Studierenden der Hardwarekatalog zur Verfügung. Er stellt eine Auswahl von oft verwendeten Komponenten in den vergangenen Projekten dar und bietet Hilfe in der Montage und Ansteuerung der Komponente.

Zu jeder Komponente gibt es eine passende Java-Klasse, welche ein einfaches Interface für die Ansteuerung bereitstellt. Zudem stellt der Katalog den Code für eine einfache Beispielapplikation zu Verfügung.

## Event Handler

Die Ereignisbehandlung in Java ist die Prozedur, die ein Ereignis steuert und die entsprechende Aktion ausführt, wenn es auftritt. Der Code oder Satz von Anweisungen, die zur Implementierung verwendet werden, wird als Event-Handler bezeichnet. Es besteht aus zwei Hauptkomponenten: der Ereignisquelle und dem Ereignis-Listener.

## Raspberry PI Pins

Es gibt viele verschiedene Wege, die PINs zu nummerieren. Pi4J verwendet hier bevorzugt die BCM-Nummerierung. Aufgrund dieser Nummerierung kann dann entsprechend im Code angegeben werden, an welchem GPIO die Komponente hängt.

## Raspberry PI Ein- und Ausgänge

Es gibt verschiedene Funktionen für die GPIO. Beispielsweise digitale Inputs / Outputs, I2C, SPI oder PWM. Jede davon kann auf verschiedene Komponenten angewendet werden.

## Task 9/9: Validierung

### Aufgabe

Kreuze an, was du verstanden hast.

## Task 1/8: Übersicht

Diese Lektion gibt eine Einführung in Klassen und Strukturen in Java. Der Schwerpunkt liegt dabei auf der applikationspezifischen Modifizierung einer bestehenden Klasse, und wie eine neue Klasse aus bereits bestehenden Klassen erstellt werden kann.

### Lernziele

- Ich kenne den Umgang mit Klassen und Strukturen in Java.
- Ich kann bestehende Klassen modifizieren.
- Ich kann bestehende Klassen in einer neuen Klasse wiederverwenden.

### Zeitschätzung

Diese Lektion wird etwa 80 Minuten in Anspruch nehmen.

### Aufbau

Der Aufbau dieser Lektion sieht wie folgt aus:

- Übersicht - *Theorie*
- Selbsteinschätzung - *MultipleChoice*
- Klassenmodifizierung - *Theorie*
- Implementation Klassenmodifizierung - *Educational*
- Wiederverwendbarkeit - *Theorie*
- Implementation Klassenwiederverwendung - *Educational*
- Zusammenfassung - *Theorie*
- Validierung - *MultipleChoice*

## Task 2/8: Selbsteinschätzung

Die Selbsteinschätzung dient dazu, zu klären, ob bereits ausreichend Wissen zu den Themen der Lektion vorhanden ist, damit diese übersprungen werden kann.

Wenn alle unten aufgelisteten Multiple-Choice-Fragen beim ersten Mal richtig beantwortet werden, kann diese Lektion mit dem Link übersprungen werden. Bei Unklarheiten sind die korrekten Lösungen als *Hint* hinterlegt.

Ja, ein Objekt (Java-Klasse) besteht aus Daten, Verhalten und dem Konstruktor.

Nein, die Klasse ist der Bauplan, wohingegen ein Objekt eine Instanz vom Typ einer bestimmten Klasse ist.

Ja, dies wird häufig verwendet, um Standardparameter trotzdem überschreiben zu können.

Ja

Nein, mit Encapsulation kann der Zustand eines Objektes vor externen Zugriffen geschützt werden.

[Wenn alles verstanden wurde, kann diese Lektion übersprungen werden](#), ansonsten kann die Lektion mit dem Button *Next* gestartet werden.

## Task 3/8: Klassenmodifizierung

Bestehende Klassen können als Vorlage verwendet werden, um neue Klassen zu erstellen. Dazu kann die bestehende Klasse einfach kopiert und notwendige Änderungen vorgenommen werden.

Der Foliensatz [Klassen und Strukturen](#) bietet eine gute Einführung in die Theorie von Klassen und Strukturen.

## Task 4/8: Implementaion Klassenmodifizierung

In dieser Aufgabe geht es darum, aus der vorhandenen Klasse [SimpleButton](#) (aus dem Hardwarekatalog) eine neue Klasse *Switch* zu erstellen. Ein Button hat die Zustände *Eingeschaltet*, *Ausgeschaltet* und *aktuell eingeschaltet*. Der Switch soll nur noch die beiden Zustände *Eingeschaltet* und *Ausgeschaltet* haben.

### Hinweis:

Die Klasse *SimpleButton* wurde bereits aus dem Hardwarekatalog kopiert und in den Ordner src eingefügt.

## Aufgabe

Erstelle die Klasse *Switch* und teste diese auf dem Raspberry Pi.

### Programmierung

- Kopiere den Inhalt der Klasse *SimpleButton* in die Klasse *Switch*
- Lösche alle Kommentare
- Lösche alle Attribute und Methoden, die etwas mit *WhilePressed* zu tun haben
- Deklariere, Initialisiere und registriere Funktionen für den *Switch*
- Deregistriere alle Funktionen des *Switches*

**Hinweis:** Die Aufgabe soll in den Klassen Main und *Switch* gelöst werden. Vergiss nicht, die Konstruktoren sollten nicht mehr *SimpleButton*, sondern *Switch* heißen. Auch im Konstruktor hat es noch *WhilePressed* Referenzen.

Der Name *switch* ist bereits von Java Funktionen belegt. Somit muss ein anderer Name für den *Switch* beim deklarieren gewählt werden. Die *Main.java* könnte wie folgt aussehen:

```
// Run the application
System.out.println("Application is running");

// Create a new Switch component
Switch obj = new Switch(pi4j, PIN.D26, false);

// Register functions to the States of the switch
obj.onDown(() -> System.out.println("XXX"));
obj.onUp(() -> System.out.println("YYY"));

// Running the App for 15 Seconds
sleep(15000);

// DeRegistering functions before shutting down
obj.deRegisterAll();

// End of application
System.out.println("Application is done");
```

### Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

### Aufbau

Als Nächstes muss der *Switch* mit dem Raspberry Pi richtig verbunden werden. Das elektrische Layout von [SimpleButton](#) zeigt eine mögliche Lösung. Für den Testaufbau können als Hardwarekomponente Schalter oder Taster verwendet werden.

### Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config *Klassenmodifizierung-ImplementationKlassenmodifizierung* hinterlegt.

## Task 5/8: Wiederverwendbarkeit

Klassen können in anderen Klassen wiederverwendet werden.

### Klassen wiederverwenden

Eine effektive Methode, eine neue Klasse für ein Bauteil zu schreiben, ist zu klären, ob es Klassen gibt, welche einzelne Grundfunktionalitäten des Bauteiles bereits abdecken. Wird eine Klasse LedButton benötigt, liegt es nahe, Methoden aus den Klassen *SimpleLed* und *SimpleButton* zu verwenden, um die gewünschten Methoden zu realisieren.

Um zum Beispiel die LED von *LedButton* mit einer Methode *ledOn()* einzuschalten muss in dieser Methode nur die Methode von *SimpleLed on()* aufgerufen werden.

## Task 6/8: Implementation Klassenwiederverwendung

In dieser Aufgabe geht es darum, aus den vorhandenen Klassen *SimpleButton* und *SimpleLed* (aus dem Hardwarekatalog) eine neue Klasse LedButton selbst zu erstellen. Die neue Komponente soll die zwei vorhandenen Objekte mit ihren Methoden als eine eigene neue Komponente zusammenfügen.

#### Hinweis:

Die Klassen *SimpleButton* und *SimpleLed* wurden bereits aus dem Hardwarekatalog kopiert und in den Ordner src eingefügt. Eigener Code muss nur noch in der Klasse LedButton geschrieben werden.

### Aufgabe

#### Programmierung

##### Main.java

Die Funktionen ledButton sind auskommentiert, damit die Übersetzung funktioniert. Kommentiere alle Funktionen (6 Zeilen) wieder ein.

##### LedButton.java

- Deklariere zwei Objekte led und button mit SimpleLed und SimpleButton
- Initialisiere die beiden Objekte im Konstruktor von LedButton
- Schreibe die unten erwähnten Methoden. Verwende dazu die Methoden von *led*.
  - ledSetState(boolean)
  - ledOn()
  - ledOff()
  - ledToggleState()
  - ledGetDigitalOutput()
- Schreibe die unten erwähnten Methoden. Verwende dazu die Methoden des instanzierten *SimpleButton*.
  - btnGetState()
  - btnIsDown()
  - btnIsUp()
  - btnGetDigitalInput()
  - btnOnDown(Runnable)
  - btnOnUp(Runnable)
  - btnWhilePressed(Runnable)
  - btnDeRegisterAll()
  - btnGetOnUp()
  - btnGetOnDown()
  - btnGetWhilePressed()

#### Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

#### Aufbau

Als Nächstes muss der LedButton mit dem Raspberry Pi richtig verbunden werden. Nutze dazu die Dokumentation [\*LedButton\*](#) des Hardwarekataloges auf der Pi4J Webseite.

#### Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config *Klassenmodifizierung-ImplementationKlassenwiederverwendung* hinterlegt.

## Task 7/8: Zusammenfassung

Dieser Task beinhaltet eine kurze Zusammenfassung, was in dieser Lektion alles abgedeckt wurde.

### Klassen / Strukturen

Klassen sind Baupläne für Objekte. Objekte bestehen aus Attributen, Konstruktoren und Methoden. Die Struktur von Objekten sollte immer die gleiche sein, beginnend mit den Attributen, gefolgt von den Konstruktoren und Methoden.

### Klassen modifizieren / wiederverwenden

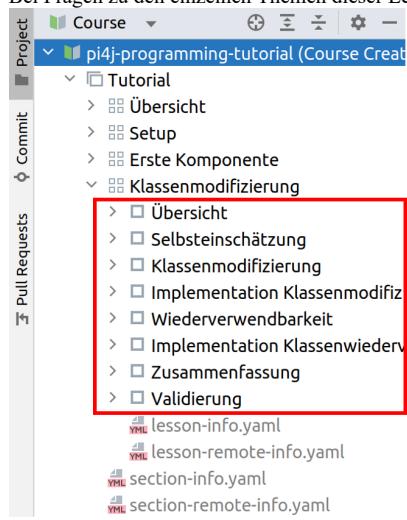
Für neue Klassen können häufig Codeelemente aus bestehenden Klassen kopiert und angepasst werden. Diese Technik führt zu einem schnellen Ergebnis, aber auch zu viel dupliziertem Code, was sich wiederum auf Qualität des Codes auswirkt. Eine bessere Lösung ist, bestehende Klassen in einer neuen Klasse wiederzuverwenden. Somit können neue Klassen auf die speziellen Eigenschaften von Bauteilen zugeschnitten werden, ohne die Applikation mit unnötigem Code zu vergrößern.

## Task 8/8: Validierung

### Aufgabe

Kreuze an, was du verstanden hast.

Bei Fragen zu den einzelnen Themen dieser Lektion kann das gewünschte Kapitel im Projektbaum geöffnet und nochmals repetiert werden.



## Task 1/10: Übersicht

Diese Lektion führt in die PI4J Umgebung ein. Sie erklärt, was ein Context ist, wie man diesen richtig initialisiert, was Provider sind und wie diese verwendet werden, und weshalb ein Context immer richtig heruntergefahren werden sollte.

### Lernziele

- Ich weiss was ein PI4J Context ist.
- Ich kenne die Provider.
- Ich verstehe, weshalb der Context heruntergefahren werden sollte.
- Ich weiss wann ich welche Provider nutzen sollte.
- Ich kann den Context selbst initialisieren.

## Zeitschätzung

Diese Lektion wird etwa 90 Minuten in Anspruch nehmen.

## Aufbau

Der Aufbau dieser Lektion sieht wie folgt aus:

- Übersicht - *Theorie*
- Selbsteinschätzung - *MultipleChoice*
- Einleitung - *Theorie*
- Standardcontext - *Edu*
- Provider - *Theorie*
- Providerwahl - *MultipleChoice*
- Shutdown - *Theorie*
- Shutdownimplementation - *Edu*
- Zusammenfassung - *Theorie*
- Validierung - *MultipleChoice*

## Task 2/10: Selbsteinschätzung

Die Selbsteinschätzung dient dazu, zu klären, ob bereits ausreichend Wissen zu den Themen der Lektion vorhanden ist, damit diese übersprungen werden kann.

Wenn alle unten aufgelisteten Multiple-Choice-Fragen beim ersten Mal richtig beantwortet werden, kann diese Lektion mit dem Link übersprungen werden. Bei Unklarheiten sind die korrekten Lösungen als *Hint* hinterlegt.

Ja, piGpio initialisiert den Provider.

Nein, es gibt einen automatischen Kontext. Den newAutoContext().

Ja, momentan hat pi4j genau 2 Provider, den piGpio und den LinuxFS.

Ja.

Nein, das Löschen und das Ausschalten geben die Ressourcen frei.

[Wenn alles verstanden wurde, kann diese Lektion übersprungen werden](#), ansonsten kann die Lektion mit dem Button *Next* gestartet werden.

## Task 3/10: Einleitung

### PI4J Context

Hier die [Theorie](#).

### Fallbeispiel

Im Beispiel des SourceCodes wird selbst ein Context erstellt, welcher alle verschiedenen GPIO-Handlers vom Provider PiGPIO verwendet. Dies könnte auch abgekürzt werden, und nur die Komponenten verwendet werden, die für die Applikation wirklich gebraucht werden.

## Task 4/10: Standardcontext

In dieser Aufgabe geht es darum, den Context einer kleinen Applikation mit den minimal benötigten Providern selbst zu schreiben. Das Gerüst ist eine Applikation, welche die Interaktion mit einer simplen LED ermöglicht.

#### *Hinweis:*

Die Klasse *SimpleLed* wurde bereits aus dem Hardwarekatalog kopiert und in den Ordner src eingefügt.

## Aufgabe

### Programmierung

- Erstelle den Context.

---

**Hinweis:** Die Aufgabe kann nur in der Klasse Main.java gelöst werden. Der Platzhalter zeigt an, wo in der SimpleLed ist ersichtlich, was für I/O Funktionalitäten die Klasse braucht.

---

## Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

## Aufbau

Als Nächstes muss die LED mit dem Raspberry Pi richtig verbunden werden. Das elektrische Layout von [SimpleLed](#) zeigt eine mögliche Lösung.

## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config [Einführung-PI4J-Standardcontext](#) hinterlegt.

## Task 5/10: Provider

### PI4J Provider

Hier die [Theorie](#).

## Task 6/10: Providerwahl

### Aufgabe

Welcher I/O Provider sollte laut Hardware-Katalog von LinuxFS genutzt werden, anstatt von PiGPIO?

Die I2C Funktionen sollten vom Provider LinuxFS genutzt werden, da der PiGPIO-I2C Provider nicht sauber in die Register der Komponenten schreiben kann. PiGPIO hat Probleme mit dem Timing des I2C. Dies kann zu Problemen mit dem LCD-Display führen.

## Task 7/10: Shutdown

### Shutdown

Das Beenden / Zerstören des Kontexts stoppt und gibt alle Ressourcen, Threads, Listener und bereitgestellten I/O-Instanzen frei, die vom Kontext gehalten werden. Dies ist nützlich, wenn bspw. mehrere Kontexte für mehrere Komponenten aufgebaut werden, und sich diese nicht in den Weg kommen. So kann jeweils ein Kontext heruntergefahren werden, bevor der andere instanziert / gestartet wird.

## Task 8/10: Shutdown Implementation

In dieser Aufgabe geht es darum, den Context und den Shutdown einer kleinen Applikation mit den minimal benötigten Providern selbst zu schreiben. Das Gerüst ist eine Applikation, welche die Interaktion mit einem simplen Button ermöglicht.

---

**Hinweis:**

Die Klasse *SimpleButton* wurde bereits aus dem Hardwarekatalog kopiert und in den Ordner src eingefügt.

---

# Aufgabe

## Programmierung

- Erstelle den Context.
- Deregistriere die Funktionen auf dem Button.
- Fahre den Kontext herunter.

**Hinweis:** Die Aufgabe kann nur in der Klasse Main.java gelöst werden. Die Platzhalter zeigen die Stelle im Code an, die anpassungen benötigen. Im SimpleButton ist ersichtlich, was für I/O Funktionalitäten die Klasse braucht.

## Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

## Aufbau

Als Nächstes muss die LED mit dem Raspberry Pi richtig verbunden werden. Das elektrische Layout von [SimpleButton](#) zeigt eine mögliche Lösung.

## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Raspberry Pi ausgeführt zu werden. Dazu ist eine Running-Config [Einführung-Pi4J-Shutdownimplementation](#) hinterlegt.

# Task 9/10: Zusammenfassung

Dieser Task beinhaltet eine kurze Zusammenfassung, was in dieser Lektion alles abgedeckt wurde.

## PI4J Context

Der Kontext ist ein unveränderliches Laufzeitobjekt, das den konfigurierten Zustand enthält und den Lebenszyklus einer Pi4J-Instanz verwaltet. Er enthält alle geladenen Plugins, Anbieter, Plattformen, I/O-Instanzregistrierung, Umgebungskonfiguration und Laufzeitobjekte, einschließlich Executor-Thread-Pools, I/O-Ereignis-Listener usw.

## PI4J Provider

Provider sind erweiterbare Servicemodule, die für die konkrete Implementierung eines bestimmten I/O-Typs verantwortlich sind. Mehrere Anbieter für denselben I/O-Typ können gleichzeitig in einen Pi4J-Kontext geladen werden. Beispielsweise könnten ein „RaspberryPi-DigitalInputProvider“ und ein „GertBoard-DigitalInputProvider“ beide geladen werden, und beide gleichzeitig digitale Eingänge bereitstellen.

## Shutdown

Es ist wichtig, den Kontext jeweils wieder richtig herunterzufahren. So können Probleme mit Überladungen und nachträglichen Funktionsaufrufen vermieden werden.

# Task 10/10: Validierung

## Aufgabe

Kreuze an, was du verstanden hast.

# Task 1/8: Übersicht

Diese Lektion gibt eine Einführung in das ModelViewController (MVC) Design-Pattern.

## Lernziele

- Ich kenne das Design-Pattern MVC.
- Ich kenne die verschiedenen Layer (Model, View und Controller) und weiss, welche Funktionen diese übernehmen.
- Ich kann eine bestehende MVC Applikation mit eigenen Komponenten erweitern.

## Zeitschätzung

Diese Lektion wird etwa 70 Minuten in Anspruch nehmen.

## Aufbau

Der Aufbau dieser Lektion sieht wie folgt aus:

- Übersicht - *Theorie*
- Selbsteinschätzung - *MultipleChoice*
- ModelViewController - *Theorie*
- View - *Edu*
- Kontroller - *Edu*
- Model - *Edu*
- Zusammenfassung *Theorie*
- Validierung - *MultipleChoice*

## Task 2/8: Selbsteinschätzung

Die Selbsteinschätzung dient dazu, zu klären, ob bereits ausreichend Wissen zu den Themen der Lektion vorhanden ist, damit diese übersprungen werden kann.

Wenn alle, unten aufgelisteten Multiple-Choice-Fragen, beim ersten Mal richtig beantwortet werden, kann diese Lektion mit dem Link übersprungen werden. Bei Unklarheiten sind die korrekten Lösungen als *Hint* hinterlegt.

Ja, beide Views können gleichzeitig gesteuert werden, wenn Sie die gleichen Kontroller-Funktionen nutzen und beide das Model observieren.

Nein, die View beinhaltet die Komponenten.

Nein, der Kontroller verändert das Model. Die View wird vom Model benachrichtigt.

Ja, das ist der normale Ablauf.

[Wenn alles verstanden wurde, kann diese Lektion übersprungen werden](course://Tutorial/Eigenes Projekt/Übersicht/src/Main.java) ansonsten kann die Lektion mit dem Button *Next* gestartet werden.

## Task 3/8: Einleitung

### MVC Pattern

Hier die [Theorie](#) zum MVC-Pattern.

### Aufgaben der MVC-Komponenten

Das MVC Pattern unterteilt das Programm in die drei Teile: Model, View und Controller.

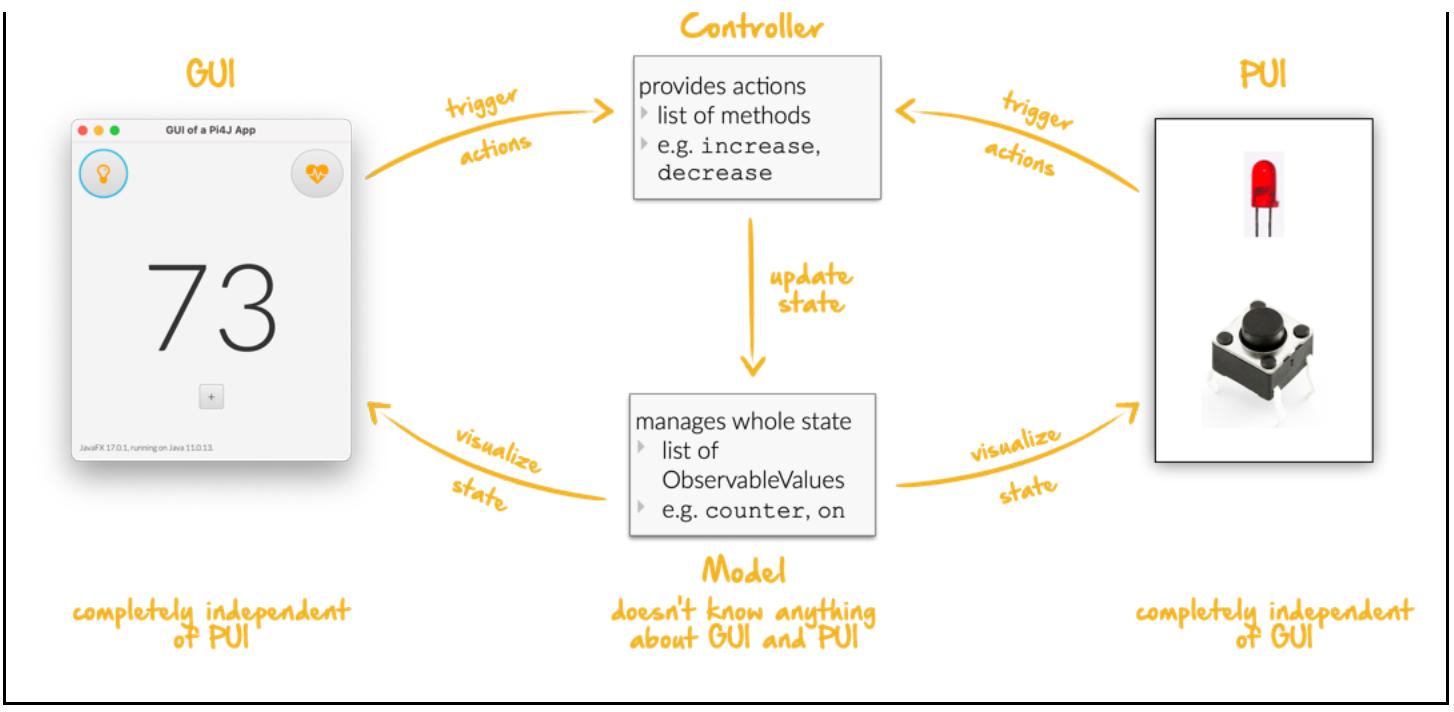
Im Model werden die Daten der Applikation gespeichert.

In der View können auf einzelne Ereignisse wie zum Beispiel *Button pressed* Methoden aus dem Controller aufgerufen werden, welche danach das Model aktualisieren. Des Weiteren wird die View verwendet, um Daten vom Model sichtbar zu machen. Zum Beispiel könnte die View einen Button haben, welcher je nach Zustand des Models gedrückt oder nicht gedrückt dargestellt wird. Es sind auch mehrere Ansichten der gleichen Information möglich. So könnte der Button gedrückt oder nicht gedrückt dargestellt und zusätzlich ein Text On/Off verwendet werden, um den Zustand nochmals zu verdeutlichen.

Der Controller stellt unterschiedliche Methoden zur Verfügung, wie die Daten im Model aktualisiert werden können. Diese Methoden können in der View auf ein bestimmtes Ereignis aufgerufen werden.

### GUI und PUI

Der Begriff Graphical User Interface (GUI) ist weit verbreitet und steht für eine Schnittstelle zwischen Benutzer (User) und Applikation. Eine LED oder ein Button sind keine grafischen, sondern physikalische Elemente. Darum wird an dieser Stelle der Begriff Physical User Interface (PUI) verwendet. Als Schnittstelle zum Benutzer sind im MVC Pattern beide Begriffe in der View angesiedelt.



## Task 4/8: View

In dieser Aufgabe geht es darum, die View in einer ModelViewController-App zu schreiben.

### Hinweis:

Die Klasse *SimpleButton* wurde bereits aus dem Hardwarekatalog kopiert und in den Ordner *view.components* eingefügt.

## Aufgabe

Um die Aufgabe zu erfüllen, muss die Klass *View.java* angepasst werden. Die Klassen *Model.java* und *Controller.java* sind im Code vorhanden, diese wurden aber für eine bessere Übersicht im Projektbau ausgeblendet.

### Programmierung

- Deklaration von *SimpleButton*
- Initialisierung von *SimpleButton* auf dem PIN 26.
- Implementation von Shutdown.
- Verbindung der Komponenten-Interaktion mit dem Kontroller.
- Ausgabe des Strings "You pressed the button x times." auf der Konsole bei Model-Änderungen.

Die Methode *onDown* des Button-Objektes kann benutzt werden, um die Funktion *PushButton* vom Kontroller aufzurufen.

### Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

### Aufbau

Als Nächstes muss der Taster mit dem Raspberry Pi richtig verbunden werden. Das elektrische Layout von *SimpleButton* zeigt eine mögliche Lösung.

### Remote Ausführung

Die Applikation ist nun bereit, um auf dem Pi ausgeführt zu werden. Dazu ist eine Running-Config *ModelViewController-View* hinterlegt. Die App wird sich selbst schliessen, wenn der Taster Zehnmal gedrückt wurde.

## Task 5/8: Kontroller

In dieser Aufgabe geht es darum, den Kontroller in der ModelViewController-App zu schreiben und die View mit einer LED zu erweitern.

#### **Hinweis:**

Die Klassen *SimpleButton* und *SimpleLed* wurden bereits aus dem Hardwarekatalog kopiert und in den Ordner *view.components* eingefügt.

## Aufgabe

Um die Aufgabe zu erfüllen, müssen die Klassen *Controller.java* und *View.java* angepasst werden.

**Hinweise:** Es existiert im Model neu die Variable *LedGlow*. Diese soll den Zustand der LED wiedergeben. TRUE bedeutet, die LED leuchtet und FALSE bedeutet, die LED ist dunkel.

## Programmierung

### Kontroller

- Auf das Ereignis Button gedrückt: aktualisiere im Model den Status von *ledGlow*.
- Auf das Ereignis Button nicht mehr gedrückt: aktualisiere im Model den Status von *ledGlow*.

Die Variablen des Models können mit *GetValue* und *SetValue* abgefragt oder geändert werden.

### View

- Deklaration von *SimpleLed*.
- Initialisation von *SimpleLed*.
- Registration der Funktion *ledOff* vom Kontroller beim Event *onUp* des Buttons.
- Mit *LedGlow* des Models die LED steuern.

Für die Registration von *ledOff* und die Ansteuerung der LED kann fast die gleiche Syntax wie für *pressButton* oder die Ausgabe des Counters verwendet werden.

## Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

## Aufbau

Als Nächstes müssen die LED und der Button mit dem Raspberry Pi richtig verbunden werden. Das elektrische Layout von *SimpleButton* und *SimpleLed* zeigen eine mögliche Lösung.

## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Raspberry Pi ausgeführt zu werden. Dazu ist eine Running-Config *ModelViewController-Kontroller* hinterlegt.

## Task 6/8: Model

In dieser Aufgabe geht es darum, das Model in der ModelViewController-App zu schreiben. Neu soll statt einer LED und einem Button direkt ein LedButton verwendet werden.

#### **Hinweis:**

Die Klassen *LedButton*, *SimpleButton* und *SimpleLed* wurden bereits aus dem Hardwarekatalog kopiert und in den Ordner *view.components* eingefügt.

## Aufgabe

Um die Aufgabe zu erfüllen, müssen die Klassen *Model.java*, *Controller.java* und *View.java* angepasst werden.

## Programmierung

- Im Model:
  - Ein Observable vom Typ String mit dem Namen *message* soll erstellt werden.
- In der View:
  - Deklaration von *LedButton*.
  - Initialisation von *LedButton*.
  - Implementation des Shutdown.
  - Verbindung der Komponenten-Interaktion mit dem Kontroller.
  - Per Aktualisierung des Models soll der neue String ausgegeben werden.
  - Per Aktualisierung des Models soll die LED gesteuert werden.
- Kontroller
  - Drücken des Buttons: Ein neuer String soll in das Model gespeichert werden.

## Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

## Aufbau

Als Nächstes muss der LEDButton mit dem Raspberry Pi richtig verbunden werden. Das elektrische Layout von [LedButton](#) zeigt eine mögliche Lösung.

## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Raspberry Pi ausgeführt zu werden. Dazu ist eine Running-Config *ModelViewController-Model* hinterlegt.

# Task 7/8: Zusammenfassung

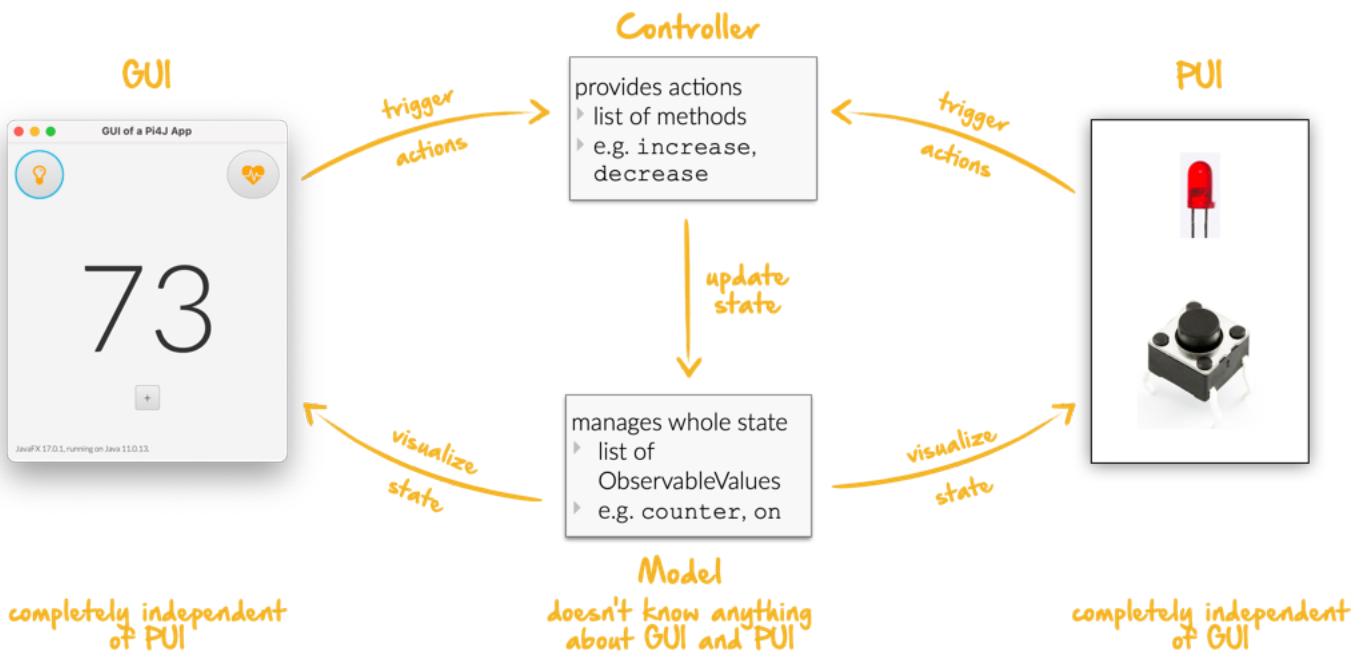
Dieser Task beinhaltet eine kurze Zusammenfassung, was in dieser Lektion alles abgedeckt wurde.

## MVC

Das MVC Pattern besteht aus dem Model, der View und dem Controller. Das Model speichert die Daten der Applikation, der Kontroller setzt auf Signale von PUI oder GUI änderungen im Model um, und die View ist zuständig für die Darstellung der Daten aus dem Model, sowie für die Bereitstellung von Interaktionsmöglichkeiten der Applikation.

## GUI und PUI

GUI steht für Graphical User Interface, während PUI für Physical User Interface steht. Das GUI besteht aus Grafischen Inhalten auf dem Bildschirm, das PUI können auch Komponenten direkt am Raspberry PI angehängt sein.



# Task 8/8: Validierung

## Aufgabe

Kreuze an, was du verstanden hast.

# Task 1/7 Übersicht

In dieser Lektion soll ein eigenes Projekt auf der Basis der TemplatePUIApp erstellt werden.

## Lernziele

- Ich kenne das TemplatePUIApp und die Struktur, wie eine App aufgebaut ist.
- Ich weiss, wie Komponenten hinzugefügt werden.
- Ich kann selbstständig ein Layout für die Hardware aufbauen.

## Zeitschätzung

Diese Lektion wird etwa 80 Minuten in Anspruch nehmen.

## Aufbau

Der Aufbau dieser Lektion sieht wie folgt aus:

- Übersicht - *Theorie*
- Einleitung - *Theorie*
- Projektstart - *Theorie*
- Schema - *MultipleChoice*
- SimonSays - *Edu*
- Zusammenfassung *Theorie*
- Validierung - *MultipleChoice*

## Task 2/7: Einleitung

### JavaFX Template

Hier die [Theorie](#) zum JavaFX-Template.

### 3 Templates

Das JavaFx-Template besteht aus 3 verschiedenen Apps sowie dem Verzeichnis für Komponenten.

#### MultiControllerApp

Die *MultiControllerApp* bietet die Möglichkeit, mehrere Controller in der gleichen Applikation zu steuern. Sie stellt eine PUI und eine GUI Klasse zur Verfügung. Es können also gleichzeitig Hardwarebauteile und grafische Objekte umgesetzt werden. Die *MultiControllerApp* ist eine App, um aufzuzeigen, wie mit mehreren Controllern gearbeitet werden kann.

#### TemplateApp

Die *TemplateApp* ist eine App, welche aus einem einzelnen Controller besteht und Implementationen für ein GUI und ein PUI bereitstellt. Dieses Template kann genutzt werden, wenn die Funktionalität der App klein ist, aber ein GUI für den Benutzer bereitgestellt wird, welches zum Beispiel durch die PUI-Komponenten gesteuert wird.

#### TemplatePUIApp

Die *TemplatePUIApp* ist vom Umfang her die kleinste App. Sie besteht aus einem einzelnen Controller und implementiert nur ein PUI. Die Applikation benötigt keine JavaFX Komponenten, sondern nur Hardware-Bauteile. Die Visualisierung wird dabei ausschliesslich durch Hardwarekomponenten realisiert.

## Task 3/7: Projektstart

### Simon Says

Simon Says soll als Memory-Spiel realisiert werden. Simon wird durch einen Computer dargestellt und schaltet 4 LEDs in unterschiedlicher Reihenfolge ein und aus. Der Spieler muss nun versuchen, die Reihenfolge wiederzugeben, so wie diese von Simon vorausgesagt wurde.

Eine mögliche Variante des Spiels ist im Editor umgesetzt. Mit dem Button *Run* kann die Demonstration gestartet werden.

### Hardware Ansatz

Diese Lektion beschäftigt sich damit, solch ein *Simon says* selbst zu implementieren und zu designen. Die wichtigsten Aufgaben dabei sind:

- elektrische Schema entwerfen
- passende Bauteile definieren
- Applikation programmieren
- Applikation testen

Die Musterlösung besteht aus 4 *LedButtons* welche in einer vom Computer zufälligen Reihenfolge aufleuchten. Diese Reihenfolge muss danach durch die Spieler exakt wiedergegeben werden. Zu Beginn einer neuen Sequenz soll jeweils auf der Console eine Meldung abgesetzt werden, und dann die Sequenz von Simon starten.

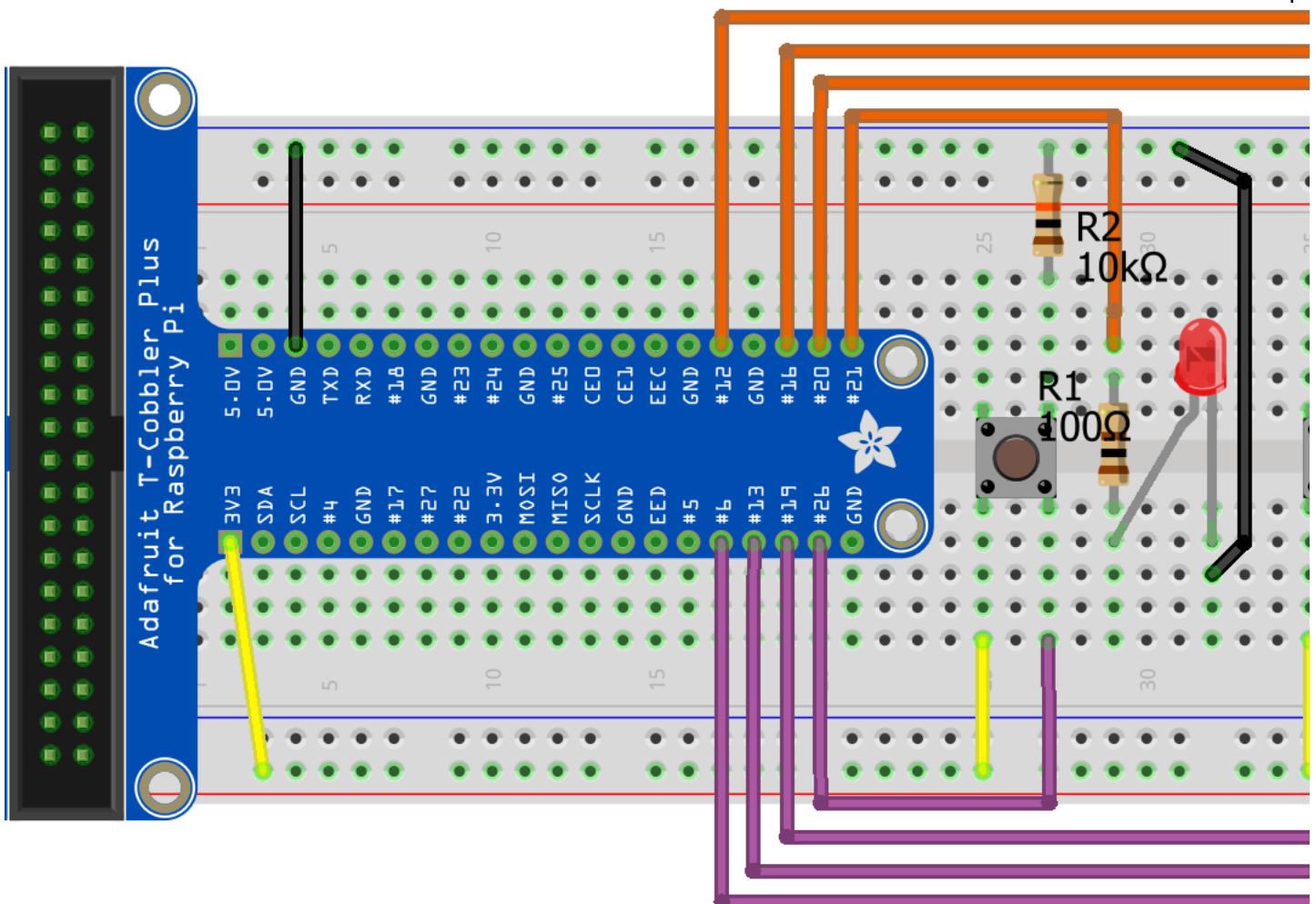
Falls die Sequenz durch die Spieler in der richtigen Reihenfolge gedrückt wird, so wird auf der Console eine Erfolgsmeldung ausgegeben. Falls in der Sequenz eine falsche Eingabe kommt, wird sofort abgebrochen und eine Fehlermeldung ausgegeben. Dann kann wieder von Vorne angefangen werden.

## Ausführung

Falls das *SimonSays* per Klick auf Run nicht ausführbar ist, muss hier wieder die RunConfiguration auf das eigene Setup angepasst werden. Diese RunConfiguration heisst *EigenesProjekt-Projektstart* und sieht etwas anders aus als wie gewohnt.

## Task 4/7: Schema

Hier wird ein Beispielschema für die vorhin genannte Applikation aufgezeigt. Es haben sich hier Fehler eingeschlichen. Markiere alle Aussagen, die im Bezug auf die vorhin genannte Applikation stimmen.



## Task 5/7: SimonSays

In dieser Aufgabe geht es darum, ein eigenes Simon Says zu erstellen. Die wichtigsten Funktionen der Applikation sind hier nochmals kurz erwähnt:

- Start der Applikation durch einen Button
- Simon gibt eine Farbfolge vor und der Spieler muss diese wiedergeben können
- Wird eine Farbfolge erfolgreich wiedergegeben, ist die nächste Farbfolge um eine Farbe länger
- Bei einem Fehler wird das Spiel beendet, mit einem Taster kann danach ein neues Spiel gestartet werden

## Model

Im ersten Schritt geht es darum, im Model die benötigten Variablen zu definieren.

Als Hilfestellung sind alle Variablen aufgelistet, welche in der Musterlösung verwendet werden.

*ledsGlowing* vom Typ BooleanArray -> Zustand der LEDs

*message* vom Typ String -> String um Mitteilungen an den Spieler über die Kommandozeile oder ein Display auszugeben

## Kontroller

Als Nächstes müssen im Kontroller sämtliche Methoden erstellt werden, welche benötigt werden, um das Modell zu aktualisieren.

Ein möglicher Lösungsansatz ist es, das Spiel in zwei Teile, Sequenz zeigen und Sequenz abfragen, zu unterteilen.

Um dies zu erreichen, werden auch Variablen benötigt, welche nicht von der View dargestellt werden. Diese können direkt im Kontroller erfasst werden.

```
//possible variables
private Integer level = 0;
private Integer startNumberOfLed = 4;
private ArrayList<> sequence = new ArrayList<>(List.of(0,1,2,3,0,0,0,0,0));
private Integer numberOfPressedLed = 0;
```

### Sequenz zeigen

Zuerst soll eine zufällige Sequenz generiert werden.

```
//create random sequence
for (int i = 0; i < get(model.sequence).size(); i++){
    Random random = new Random();
    get(model.sequence).set(i, random.nextInt(4));
}
```

Danach sollen die LED entsprechend ein- und ausgeschaltet werden.

```
for (int i = 0; i < (startNumberOfLed+level-1); i++){
    int currentLed = get(model.sequence).get(i);
    setValue(model.led0IsGlowing,true);
    sleep(1000);
    setValue(model.led0IsGlowing,false);
    sleep(500);
}
```

### Sequenz abfragen

Bei jedem Button der gedrückt wird, muss kontrolliert werden, ob die Sequenz noch stimmt. Zusätzlich müssen noch weitere Fragen geklärt werden:

- ist das nächste Level erreicht
- ist das Spiel beendet

Überprüfe, ob die richtige Sequenz eingehalten wird

```
//check if right button is pressed
if(btnNumber == sequence.get(numberOfPressedLed)){
    increase(model.numberOfPressedLed);
    sleep(100);
    setValue(model.message, "Your " + numberOfPressedLed + " button was button number " + btnNumber + ". This was right.");
} else{
    setValue(model.message, "You pressed the wrong button. press any button to restart the game");
    level = 0;
    return;
}
```

Überprüfe, ob das Level oder das Spiel beendet ist

```
//check if level is completed
if(numberOfPressedLed >= startNumberOfLed+level-1){
    setValue(model.message, "You have completed level" + level);
    level++;
    //check if game is completed
    if(level+startNumberOfLed > sequence.size()){
        setValue(model.message, "You have finished the game, press any button to restart the game");
        level=0;
        return;
    }
    numberOfPressedLed = 0;
    showNewSequence();
}
```

## View

In der View geht es nun darum, die LED Buttons richtig zu implementieren.

Hier ist eine Hilfsstellung für die ModelToUIBindigns

```
onChangeOf(model.ledsGlowing).execute((oldValue, newValue) -> {
    for (int i = 0; i < ledButtons.length; i++) {
        ledButtons[i].ledSetState(newValue[i]);
    }
});

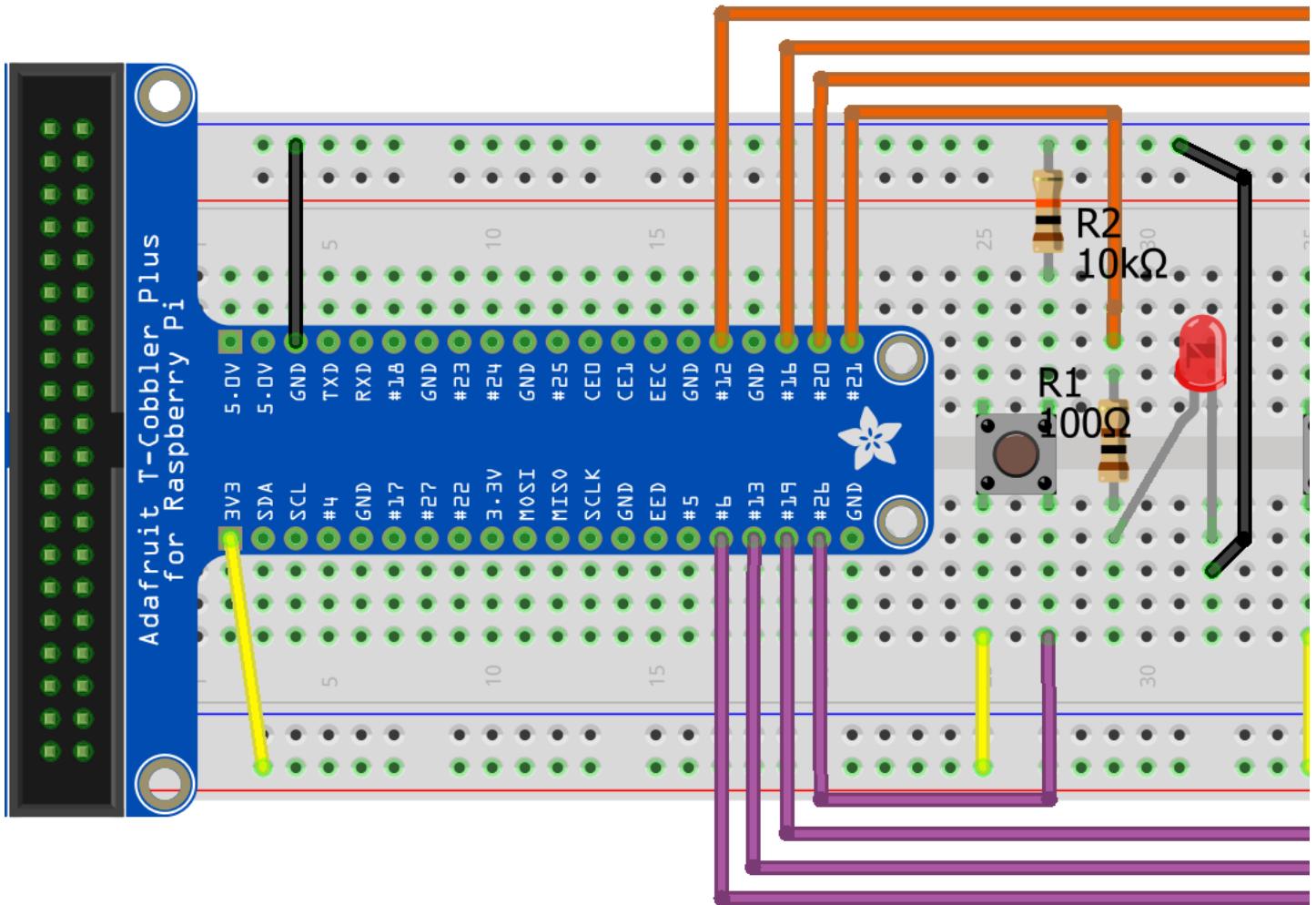
onChangeOf(model.message).execute((oldValue, newValue) -> System.out.println(newValue));
```

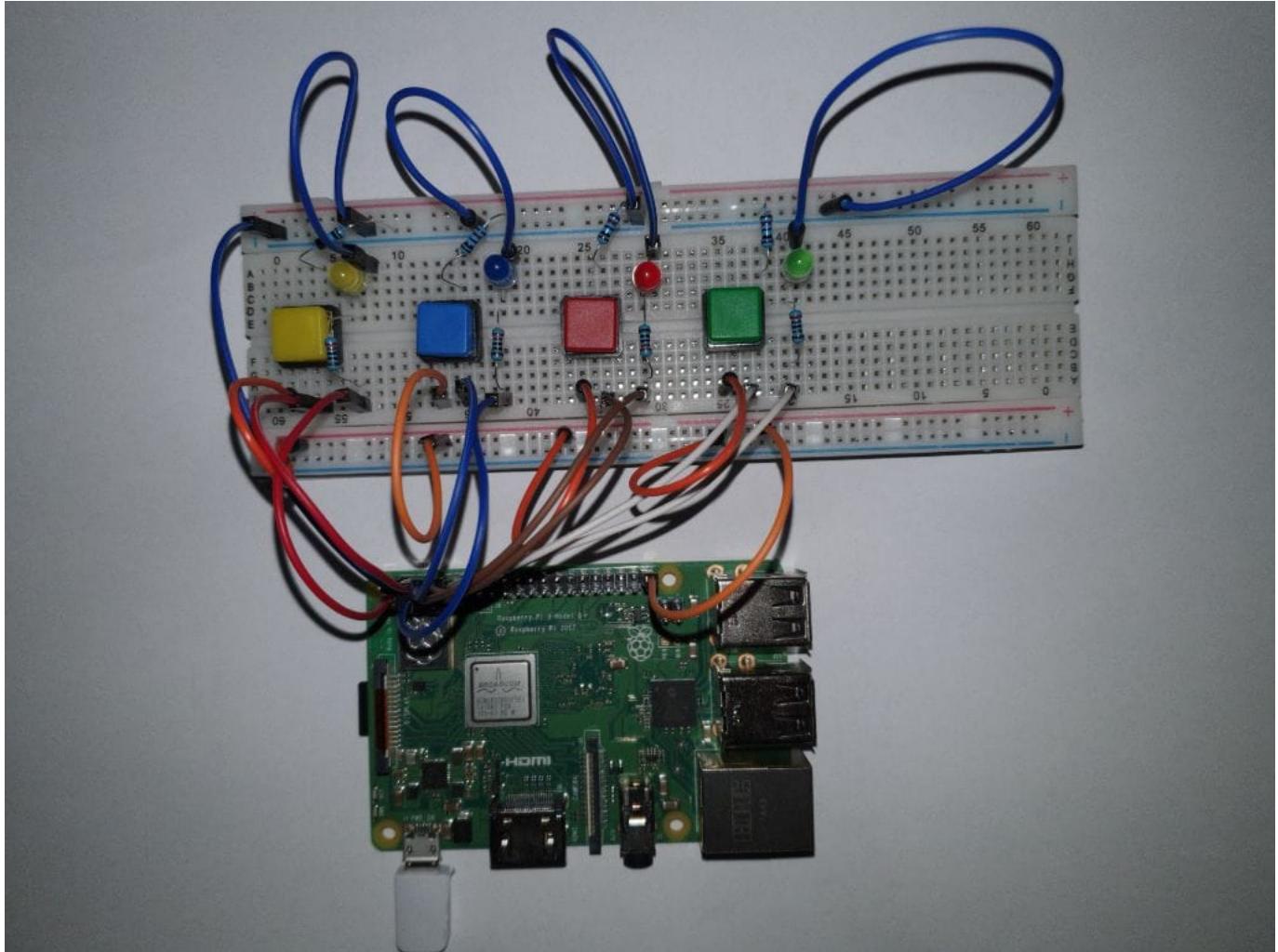
## Check Programmierung

Die Programmierung lässt sich mit dem Button *Check* überprüfen. Nach erfolgreich abgeschlossenem Test kann mit der Erstellung der Hardware begonnen werden. Ist der Test nicht erfolgreich, muss zuerst der Fehler behoben werden. Dabei können allfällige Hints oder ein Blick in die Lösung helfen.

## Aufbau

Als Nächstes muss die gewählte Hardware mit dem Raspberry Pi richtig verbunden werden. Die Musterlösung mit 4 *LEDButtons* könnte wie folgt aussehen:





## Remote Ausführung

Die Applikation ist nun bereit, um auf dem Raspberry Pi ausgeführt zu werden. Dazu ist eine Running-Config *EigenesProjekt-Komponenten* hinterlegt.

## Task 6/7: Zusammenfassung

Dieser Task beinhaltet eine kurze Zusammenfassung, was in dieser Lektion alles abgedeckt wurde.

### JAVAFXApp

Die JavaFXApp ist eine Vorlage zur Verwendung der MVC-Architektur zur Implementierung einer JavaFX-basierten Applikation. Die App besteht aus 3 Vorlagen.

### SimonSays im Hardware-Ansatz

In dieser Lektion musste selbst ein SimonSays implementiert werden. Mit 4 LED, welche an digitalen Ausgängen des Raspberry Pi angeschlossen sind, wird eine Farbfolge vorgegeben, die danach durch die Spieler über 4 Buttons, die mit digitalen Eingängen verbunden sind, wiederholt werden muss.

## Task 7/7: Validierung

### Aufgabe

Kreuze an, was du verstanden hast.