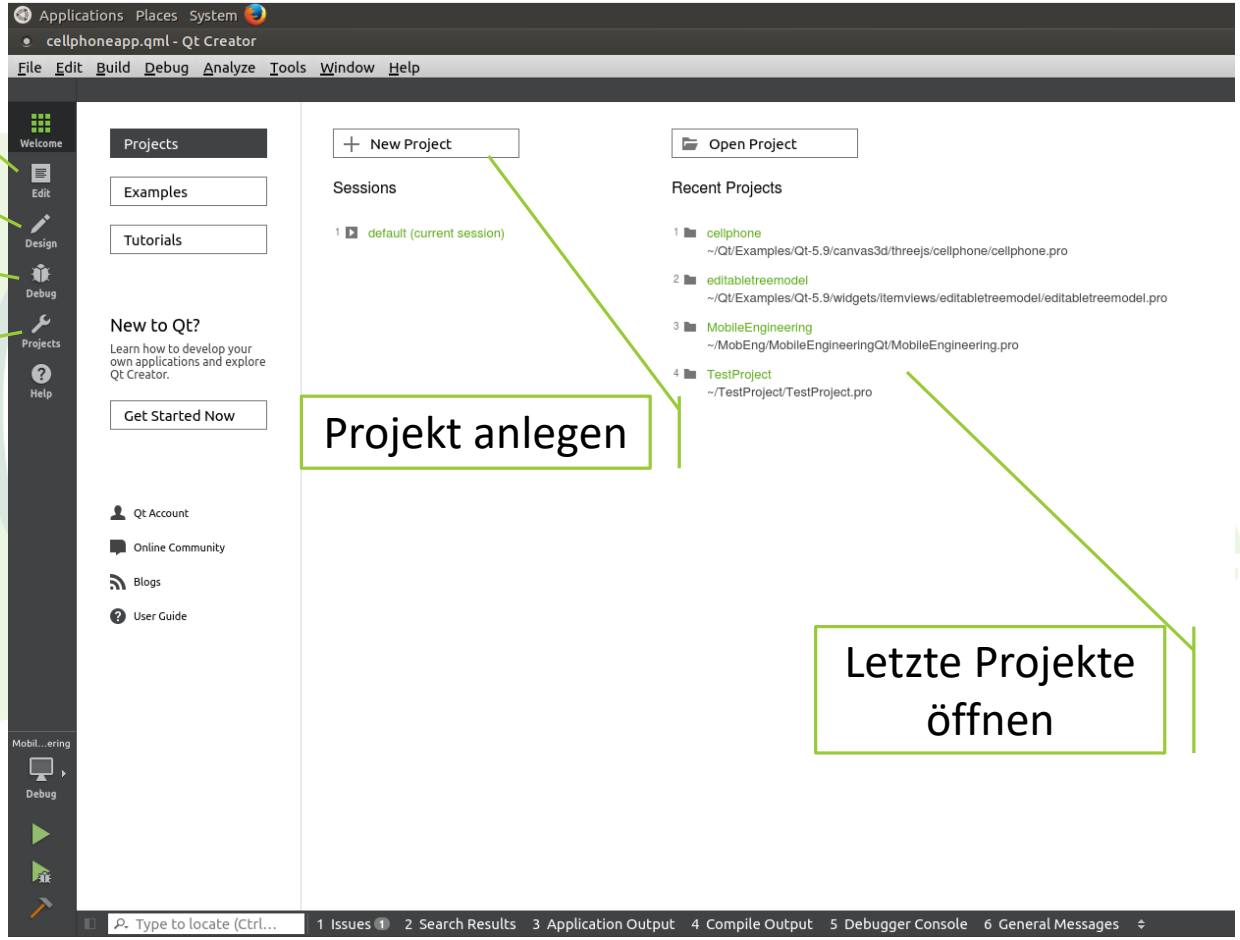
Laboraufgabe Qt

HENNER BENDIG

SVEN OLE LUX

04.04.2017

QtCreator



Quellcodes

GUI Designer

Debugger

Projekt-einstellungen

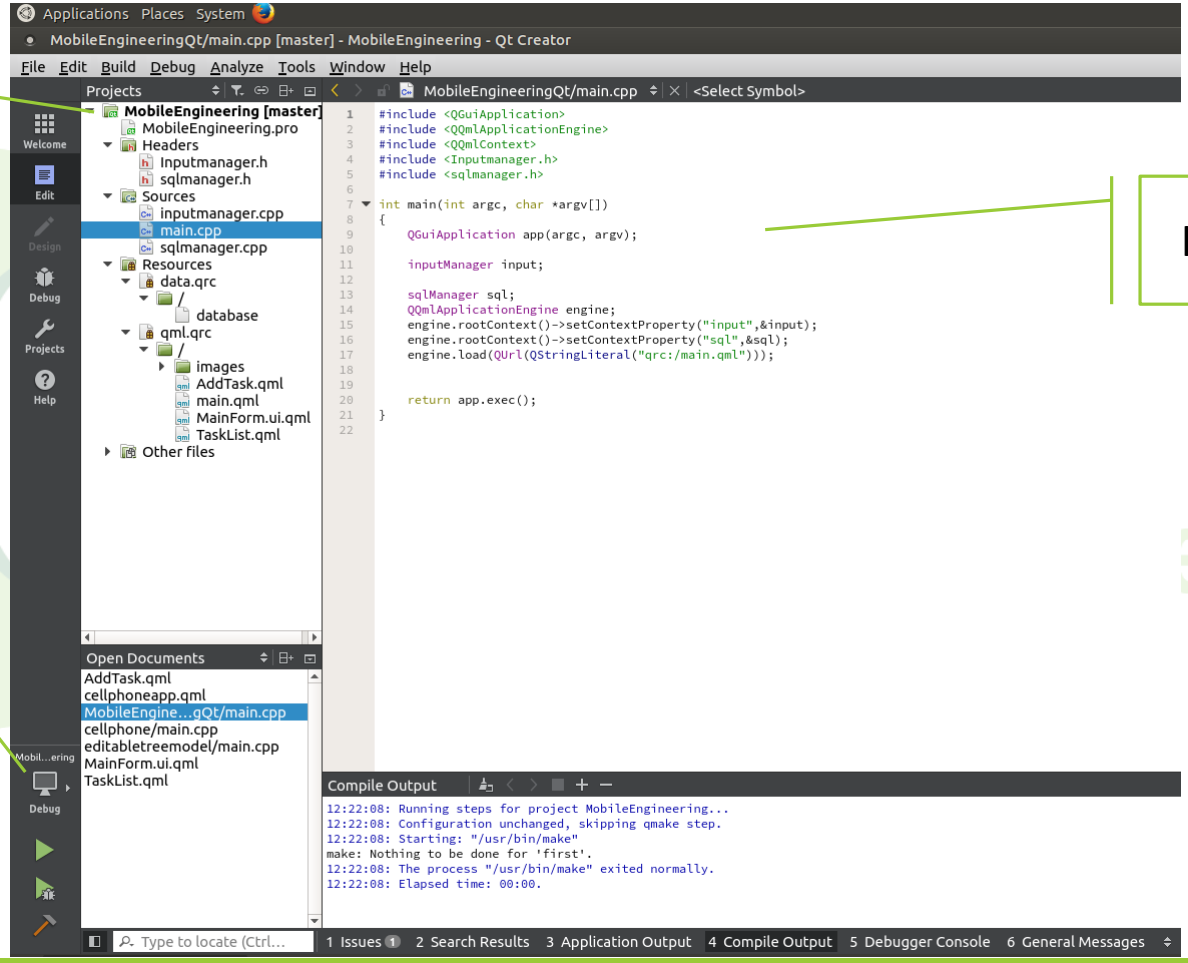
Projekt anlegen

Letzte Projekte öffnen

QtCreator

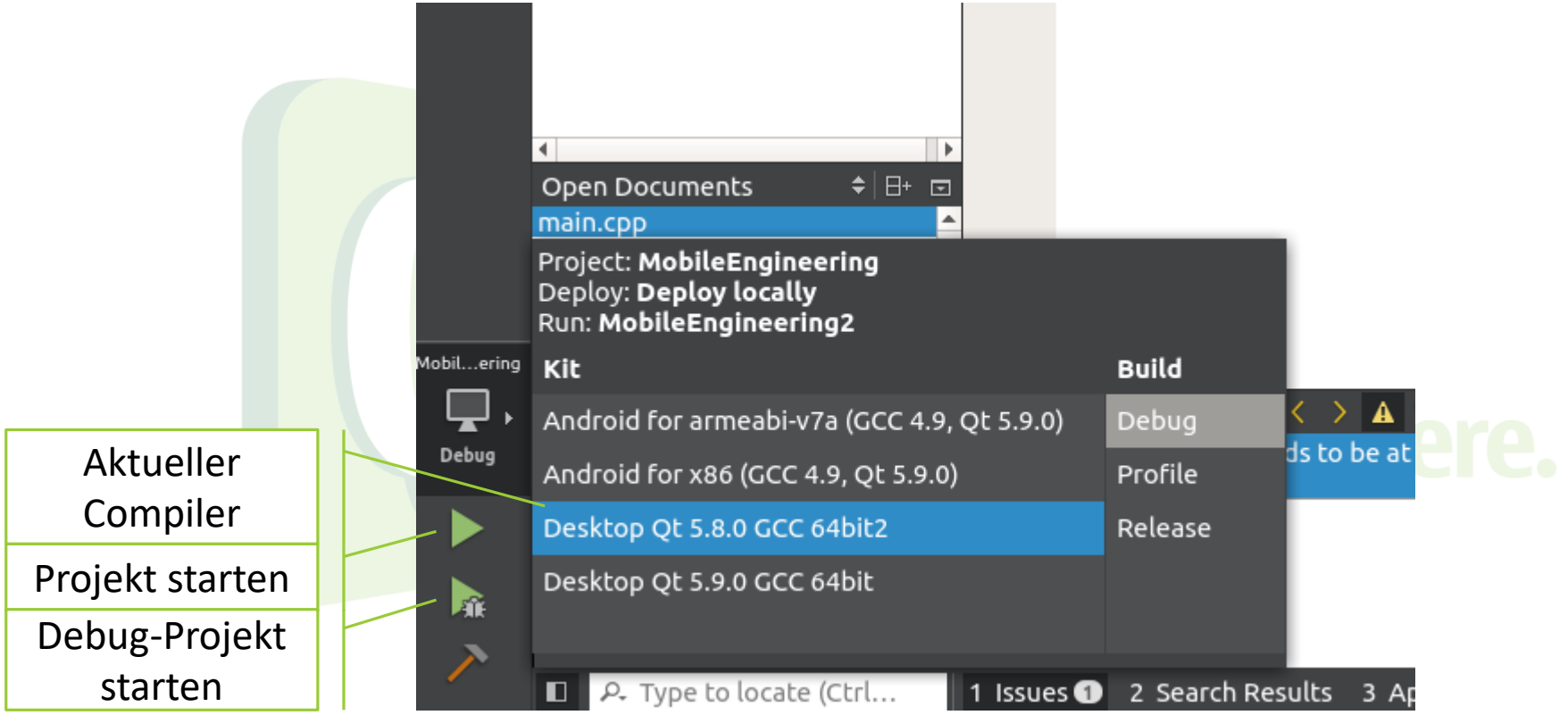
Aktives Projekt

Compiler-auswahl



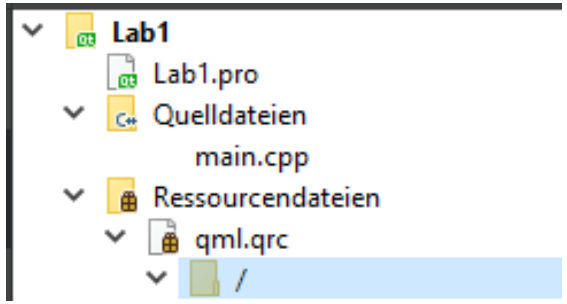
Hier coden ...

QtCreator


Aktueller Compiler
Projekt starten
Debug-Projekt starten
Open Documents
main.cpp
Project: MobileEngineering
Deploy: Deploy locally
Run: MobileEngineering2
Kit
Build
Android for armeabi-v7a (GCC 4.9, Qt 5.9.0)
Android for x86 (GCC 4.9, Qt 5.9.0)
Desktop Qt 5.8.0 GCC 64bit2
Desktop Qt 5.9.0 GCC 64bit
Debug
Profile
Release
Debug
Type to locate (Ctrl...
1 Issues
2 Search Results

Aufgabe Lab 1a – Hello World!

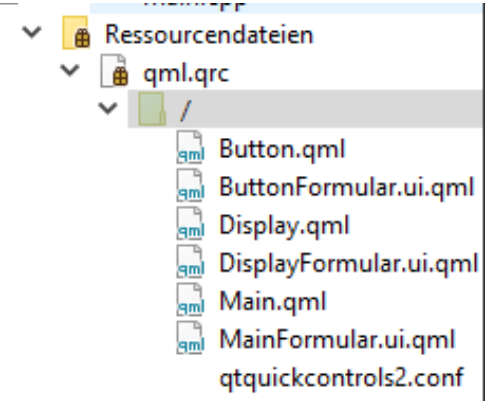
- Öffne das bestehende Projekt Lab1
- Lasse die Anwendung für den Desktop kompilieren und ausführen
  - Hinweis: Der Compiler läuft durch, aber es wird keine Anwendung erscheinen
- Füge über das Kontext-Menü „Hinzufügen“ ein neues UI-Fenster hinzu
  - Best Practise-Tipp: Klappe in der Hierachie „Ressourcendateien/qml.qrc“ auf und öffne das Kontextmenü auf dem „/“-Ordner
  - Qt > QtQuick UI-Datei, Komponentenname: Main, Komponentenformularname: MainFormular (ohne Bindestrich!!)

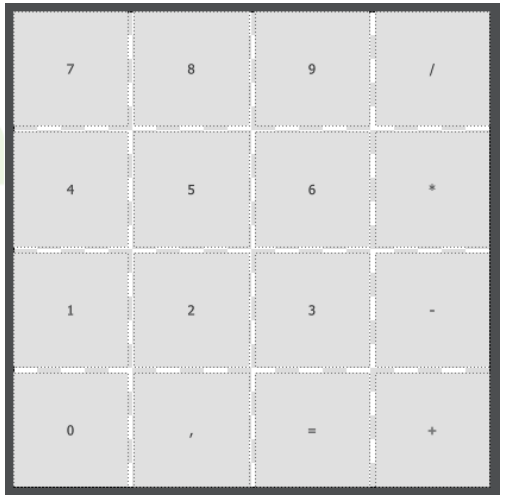


- Komponente „Main.qml“ enthält vorrangig Logik (z.B. Javascript), das Formular enthält die UI-Elemente (und kann kein JS)
- Füge in „main.qml“ ein ApplicationWindow{} hinzu, und setze es auf „visible: true“
  - Es muss hierfür QtQuick.Controls 2.1 importiert werden
  - Das MainFormular sollte innerhalb des ApplicationWindow aufgerufen werden
  - Kommentiere in der Main.cpp die Zeile 14 (`engine.load(…)`) ein
  - Wenn die Anwendung jetzt ausgeführt wird, startet ein weißes Fenster
  - Gib dem Fenster eine Höhe und Breite und füge einen Text ein
  - z.B. 400x800 mit „Hello World!“

Aufgabe Lab 1b – Calculator

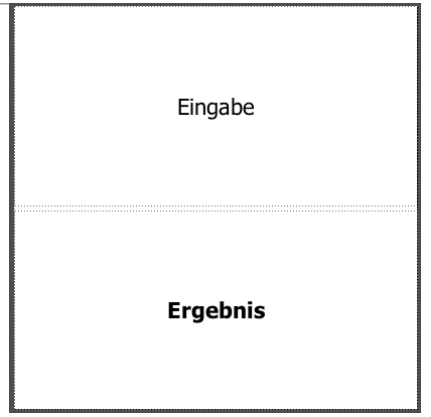
- Füge zwei weitere UI-Forms hinzu: Buttons und Display
- Passe die Formular-Dateien so an, dass:
  - Das Item des MainFormular das Eltern-Element ausfüllt
    - Siehe hierfür in die Doku: http://doc.qt.io/qt-4.8/qml-item.html, Stichwort: anchors
  - Die Items aus dem Button- und dem Display-Form 400x400px groß sind
- Dem ButtonFormular füge folgende Buttons in dieser Sortierung hinzu:
  - Tipp: Nutze GridLayout (`import QtQuick.Layouts 1.1`)
    und Controls (`import QtQuick.Controls 2.1`)
  - Die Buttons sollen sich an die entsprechende Größe anpassen



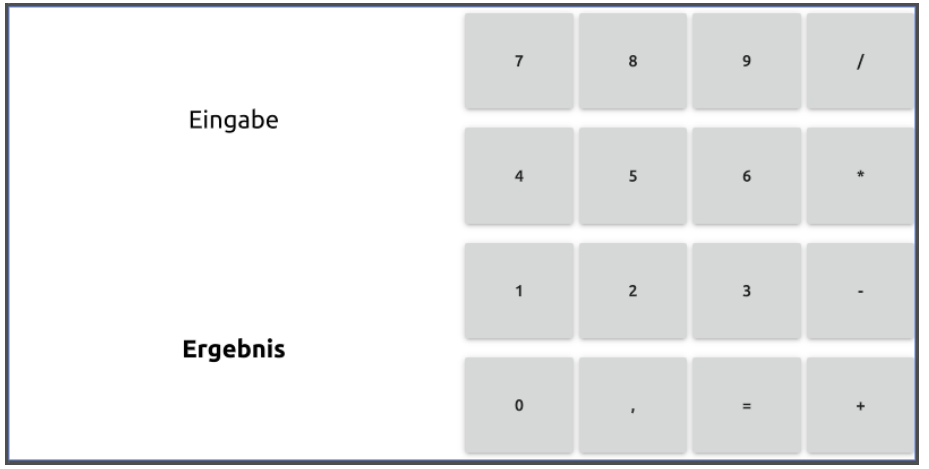


Aufgabe Lab 1c – Calculator

- Das DisplayFormular soll wie folgt aussehen:
  - Auch die Textfelder sollen sich an die zur Verfügung stehende Größe anpassen



- Füge die QML-Dateien „Display“ und „Button“ mittels Layouts zum MainFormular hinzu, dass sie nebeneinander angezeigt werden
  - Tipp: Im MainFormular erzeugst du je ein Item, dem du die Formulare (Button und Display) hinzufügst
  - Das „Hello World!“ kannst du jetzt löschen ☺



Aufgabe Lab 1d – Calculator

- Erzeuge nun eine C++-Klasse „Calculator“
  - Inkludiere die Klasse „QObject„ (`#include <QObject>`),
    lasse den Calculator von QObject erben,
    in die erste Zeile der Klasse wird das Präfix „Q_OBJECT“ geschrieben
  - Schreibe die Logik für einen Taschenrechner
    - Die Klasse Calculator soll Funktionen haben, die auf die verschiedenen Buttons reagieren
    - -> Zahl für den linken Term, Zahl für den rechten Term, Rechenoperator abspeichern,
      beim Speichern des Rechenoperators zum rechten Term wechseln, Rechenoperation ausführen, ..
    - Füge im Header „`signals: calcDone(float calcResult);`“ hinzu
    - Am Ende der Berechnung fügst du folgenden Signalaufruf hinzu: „`emit calcDone(result);`“
  - Wenn hier Hilfe benötigt wird, stellen wir die entsprechende Klasse gerne bereit ☺

```
1  #ifndef CALCULATOR_H
2  #define CALCULATOR_H
3
4  #include <QObject>
5
6  class Calculator :public QObject
7  {
8      Q_OBJECT
9  public:
10     Calculator();
```

- Inkludiere die „calculator.h“ in die main.cpp und kommentiere die entsprechende Zeile ein, um den Calculator für den QML-Context freizugeben (Zeile 11)

Aufgabe Lab 1e – Calculator

- Stelle in der ButtonFormular.ui.qml die unterschiedlichen Buttons als Eigenschaften bereit
  - http://doc.qt.io/qt-5/qtqml-syntax-objectattributes.html
  - z.B. `property alias button0: buttonIdNumber0`
- Schreibe in der Button.qml die onClicked-Methoden für die einzelnen Buttons und rufe die entsprechenden Funktionen aus der Calculator-Klasse auf
- Gebe in der DisplayFormular.ui.qml die Text-Objekte als Eigenschaften frei
- Schreibe in der Display.qml die Connection zum Calculator:
  - ```
    Connections{ target: calc
    onCalcDone: outputText = calcResult }
    ```
    ```

Aufgabe Lab 1f – Calculator

- Mit folgenden Code können die Connections zwischen den UI-Elementen aufgebaut werden:

```
DisplayFormular {

    Connections{
        target:calc
        onCalcDone: outputText.text = calcResult
    }
    Connections{
        target:mainWindow
        onClickChanged:{ inputText.text = inputText.text + mainWindow.click}
    }
    Connections{
        target:mainWindow
        onFirstClickedChanged:{
            if(mainWindow.firstClicked){
                inputText.text = ""
            }
        }
    }
    Connections{
        target:mainWindow
        onFirstClickedChanged:{
                inputText.text = ""
        }
    }
}
```

```
ApplicationWindow {
    id: mainWindow
    visible: true

    width: 800
    height: 400

    signal clicked(string clickButtonText)

    property string click : ""
    property bool firstClicked : true
    property bool awake: true

    function wasClicked(clicked){
        if(awake){
            awake = false
        }

        if(firstClicked){
            firstClicked = false
        }

        click = clicked
        console.log(click)
    }

    MainFormular {

        }
}
```

Aufgabe Lab 2 – Unterschiedliche UIs

- Lege eine zweite UI an, die beim Ausführen auf dem Smartphone (Hochformat) gestartet werden soll
  - Hier seid ihr völlig frei im Design, im einfachsten Fall können einfach die Formulare anstatt übereinander angezeigt werden, anstatt von nebeneinander
  - Die beiden Formulare (Display und Button) können wiederverwendet werden, die QML und CPP-Dateien müssen etwas angepasst werden
- In der Main.cpp kann mit Präprozessor-Macros das Betriebssystem unterschieden werden:
  - `Q_OS_ANDROID` ist `true`, wenn es ein Android-System ist
  - Weitere sind hier zu finden: http://doc.qt.io/qt-5/qtglobal.html
- Über `engine.load(QUrl(QLatin1String(„qrc://NEUEMAIN.qml“)));` Lässt sich bestimmen, welche QML geladen wird

Aufgabe Lab 3a – Sensoren

In dieser Aufgabe wird eine kleine App entwickelt, die die Nutzung der Sensoren innerhalb von Qt zeigt. Diese App kann gleichermaßen auf Android, wie auch auf iOS genutzt werden.

Als Beispiel wird eine kleine Kugel auf dem Bildschirm in Richtung der Kippung des Handys bewegt.

- Öffne das Projekt Lab 3
- In der Datei „Lab3.pro“ werden in der ersten Zeile die Module angegeben, die in das Qt-Projekt importiert werden. Es muss das Modul „sensors“ hinzugefügt werden
  - `QT += qml quick svg sensors`
- Die restliche Logik kann komplett in der „main.qml“ implementiert werden
- Importiere „QtSensors 5.0“ in die „main.qml“

Aufgabe Lab 3b – Sensoren

- In der Datei „Lab3.pro“ werden in der ersten Zeile die Module angegeben, die in das Qt-Projekt importiert werden. Es muss das Modul „sensors“ hinzugefügt werden
  - `QT += qml quick svg sensors`
- Die restliche Logik kann komplett in der „main.qml“ implementiert werden
- Importiere „`QtSensors 5.0`“ in die „main.qml“
- Der Code-Block „`Image{ … }`“ baut aus der eingebundenen SVG-Datei (/content/Bluebubble.svg) eine Kugel und fügt eine Animationsbewegung hinzu
- Füge unter „`visible: true`“ den Aufruf des Sensors hinzu, hier „`Accelerometer { }`“
  - Innerhalb der geschweiften Klammern wird nun die Logik geschrieben
  - Gib dem Accelerometer eine id, mit dieser kannst du die Daten später abfragen
  - Der Sensor wird mit dem Attribut „`active: true`“ aktiviert, außerdem muss eine Datenrate „`dataRate:`“ angegeben werden, z.B. mit einem Wert von 100

Aufgabe Lab 3c – Sensor

- Die QML-Datei sollte nun wie folgt aussehen:
- Dem Accelerometer kann nun das Event „`onReadingChanged: {}`“ hinzugefügt werden, dieses reagiert auf alle Änderungen die von dem Sensor gemeldet werden
- Mit `accelerometerID.reading.x /.y /.z` werden die Werte des Sensors abgefragt
- Mit `bubble.x / .y / .z` kann die Position der Kugel gesetzt werden
- Weitere Informationen zum Auslesen der Accelerometer-Werte:
  http://doc.qt.io/qt-5/qml-qtsensors-accelerometerreading.html

```
1  import QtQuick 2.1
2  import QtQuick.Controls 1.0
3
4  import QtSensors 5.0
5
6
7  ApplicationWindow {
8      title: "Sensor Bubble"
9      id: mainWindow
10     width: 320
11     height: 480
12     visible: true
13
14     Accelerometer {
15         id: accel
16         dataRate: 100
17         active:true
18
19     }
20
21
22     Image {
23         id: bubble
24         source: "content/Bluebubble.svg"
25         smooth: true
26         property real centerX: mainWindow.width / 2
27         property real centerY: mainWindow.height / 2
28         property real bubbleCenter: bubble.width / 2
29         x: centerX - bubbleCenter
30         y: centerY - bubbleCenter
31
32         Behavior on y {
33             SmoothedAnimation {
34                 easing.type: Easing.Linear
35                 duration: 100
36             }
37         }
38         Behavior on x {
39             SmoothedAnimation {
40                 easing.type: Easing.Linear
41                 duration: 100
42             }
43         }
44     }
45 }
46
```

Aufgabe Lab 3d – Sensoren

Folgende Funktionen helfen die Bewegung der Kugel zu berechnen:

```
43      function calcPitch(x,y,z) {
44          return -(Math.atan(y / Math.sqrt(x * x + z * z)) * 57.2957795);
45      }
46      function calcRoll(x,y,z) {
47           return -(Math.atan(x / Math.sqrt(y * y + z * z)) * 57.2957795);
48      }
```