**Ostfalia Hochschule für Angewandte Wissenschaften**

**Fakultät Informatik**

SS 2017

**Modellbasierte Codegenerierung**

**Experimental Report**

**Name, Vorname : Ndeudje Cyprez Renault , Wen Lin**

**Matrikel-Nr: 70448500 , 70452531**

**Dozent: Günter Kircher**

Inhalt

[1. Example 2 2](#_Toc484401888)

[2. Example 3 2](#_Toc484401889)

[3. Example 4 6](#_Toc484401890)

[4. MerapiUML 8](#_Toc484401891)

[4.1. Klassendiagramm 8](#_Toc484401896)

[4.2. Beschreibung Einzelne Methoden in Verschiedene Klassen 9](#_Toc484401897)

[4.3. Object Diagram dient zur Struktur des Programms 11](#_Toc484401898)

[4.4. State Diagram steht das Verfahren von dem gesamten Programm 12](#_Toc484401899)

[5. Verhältnis von Klassen 16](#_Toc484401900)

[6. XTEND Programmiersprache 19](#_Toc484401901)

[6.1. TestOperation 19](#_Toc484401902)

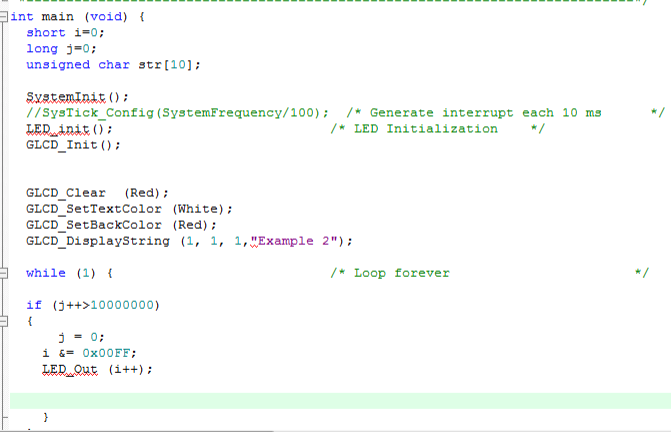
[6.2. TestParameter 21](#_Toc484401903)

[7. Aktivitäts- Diagramm 23](#_Toc484401904)

# Example 2

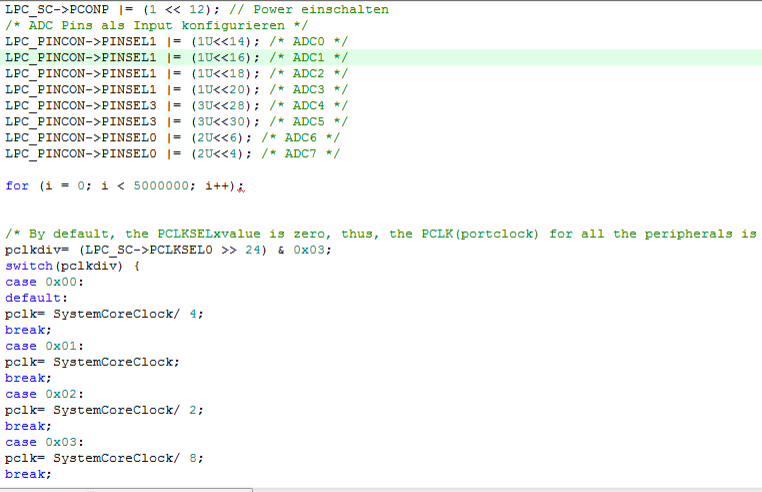
Ziel des Aufgabe war die Anzeige Verschiedene Daten auf dem Eva-Board Bildschirm und sich auch mit dem C-Programm zu gewöhnen

* Erstmal wurden das System, die LED und GLCD initialisiert
* Dann wurden die Farben konfiguriert einmal für den Hintergrund des Eva-Board Bildschirms(siehe Zeile 23) , dann für die Daten , die angezeigt werden(Zeile 24) , zuletzt für „Example 2“ (Zeile 26)

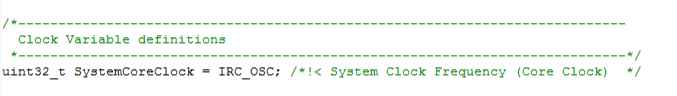


# Example 3

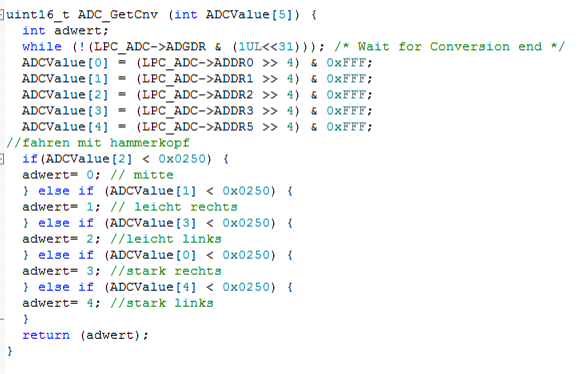
Ziel dieser Aufgabe war die Aktivation des Auto-Sensors durch erweiterung des C-Programms



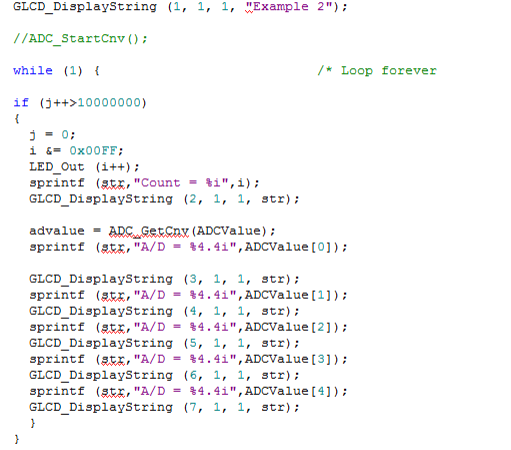
* Im ADC.c wird eine Funktion zur Initialisierung der ADC geschrieben, wobei Strom eingeschaltet wurde (Zeile 31) und dabei die ADC als Input konfiguriert(Zeile 33 bis Zeile 40)
* Dann wird in die Methode die Werte der Portclock(pclk) einzeln bestimmt(von Zeile 47 bis Zeile 60) mit SystemCoreClock ,die in system\_LPC17xx.c(Zeile 390) definiert wurde



* ADCValue wurde dann als Arrays definiert, konvertiert und die verschiedene Position des Arrays wurden dann für die Entsprechenden Richtungen(zum Beispiel Mitte, stark rechts usw.) eingerichtet und zurückgegeben. Dabei wird eine Schleife gesetzt auf die Ende der Konversion wartet(Zeile 104)

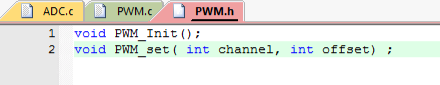


* Im Example3.c passiert es ziemlich gleich wie im Example2.c nur, dass die verschiedene Position der ADCValue für die Entsprechenden Richtungen(stark Links, leicht Rechts, Mitte usw.) zusätzlich hinzukommen(von Zeile 35 bis Zeile 56).Außerdem wurde hierbei ein Counter gesetzt, die die werten beeinflusst, wenn man vor dem Auto mit dem Straßenstückchen zu den Richtungen sich bewegt.

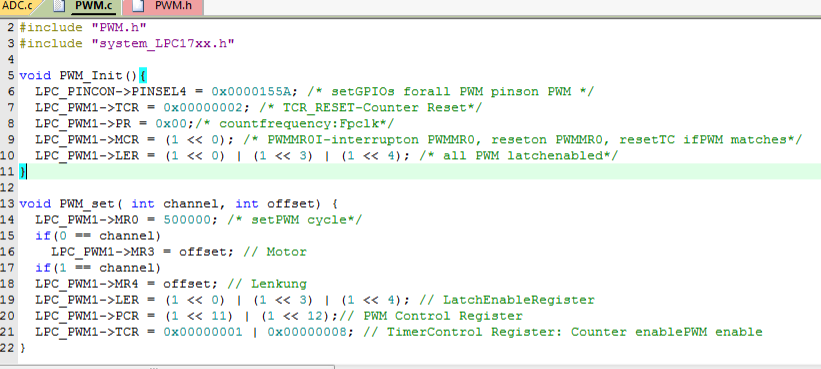


# Example 4

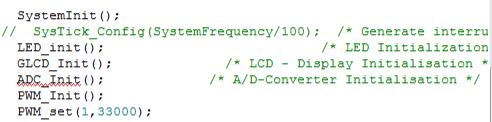
Ziel hier war die Pulsweitenmodulation(PWM) zu implementieren im Programm, die zur Lenkung und gaz geben von Auto dient



* In den PWM.h Datei wird die Methoden PWM\_Init und PWM\_set(mit zwei Parameter) deklariert
* Diese Methoden werden dann in die PWM.c Datei implementiert, wobei PWM\_Init für die Initialisierung der PWM dient (von Zeile 5 bis Zeile 10) und die PWM\_set methode für den Motor und die Lenkung am Auto dient mit den Jeweiligen Parametern channel (Motor) und Offset(Lenkung).Das ganze lässt sich dann deutlich in dem Untere Bild darstellen:



* Im example3.c wird dann die PWM\_Init und die PWM\_set Methode aufgerufen mit gewissen Parametern(1 und 33000).Dies dient, dass die Räder des Auto sich nach Links, Rechts oder in der Mitte bewegen beim Testen vorne am Sensor mit dem Straßenstückchen. In den untere Bild ist zeigt sich das ganze an:

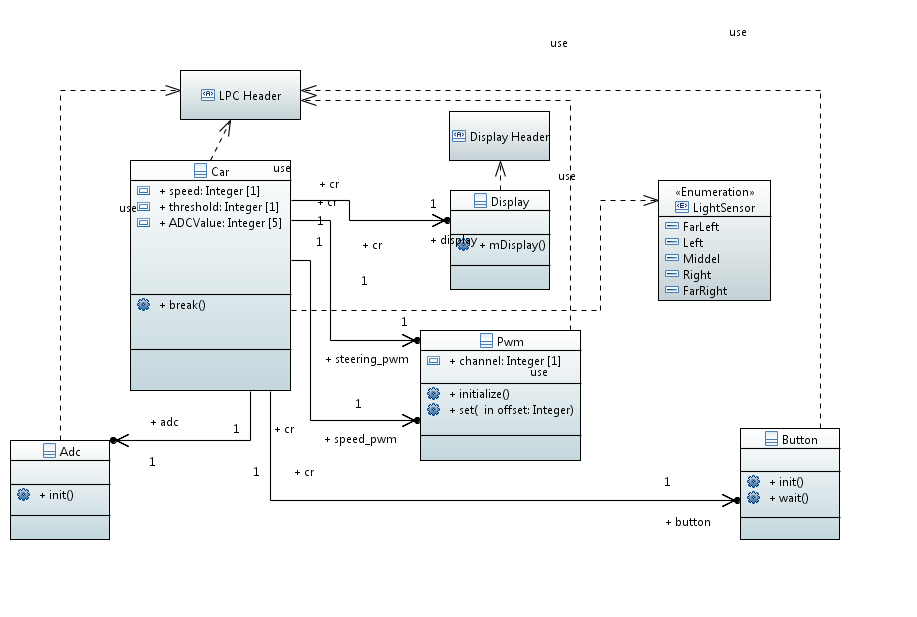


# MerapiUML

Ziel der Aufgabe hier war sich mit dem Class-,Object- und State-Diagramm und dessen Nutzung in Merapi vertraut machen und dabei codegenerieren, damit dem Auto auf eine Straße mit gegebene Geschwindigkeit fährt.



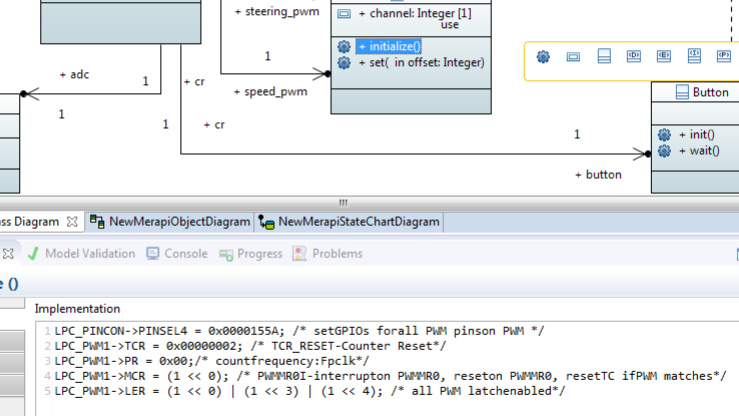
## Klassendiagramm

****

* Car ist die Hauptklasse,die die Klassen Adc,Pwm,Display und Button(beim Drücken werden sich die Räder nach einem Kligelton mit entsprechenden Geschwindigkeit drehen ,die enthält eine init- und eine wait-methode) mit jeweils Property(Attribute) und Operation(Methoden)nutzt

## Beschreibung Einzelne Methoden in Verschiedene Klassen

* Die Pwm-Klasse nutzt speed und steering.In die initialize-Methode wird die Pulsweitenmodulation initialisiert
* LPC\_PINCON->PINSEL4 = 0x0000155A; /\* setGPIOs forall PWM pinson PWM \*/
* LPC\_PWM1->TCR = 0x00000002; /\* TCR\_RESET-Counter Reset\*/
* LPC\_PWM1->PR = 0x00;/\* countfrequency:Fpclk\*/
* LPC\_PWM1->MCR = (1 << 0); /\* PWMMR0I-interrupton PWMMR0, reseton PWMMR0, resetTC ifPWM matches\*/
* LPC\_PWM1->LER = (1 << 0) | (1 << 3) | (1 << 4); /\* all PWM latchenabled\*/



* Die Pwm-Klasse nutzt speed und steering.In die set -Methode wird für die Pulsweitenmodulation eine Funktion für die Lenkung(steering) und den Motor(speed) gesetzt

LPC\_PWM1->MR0 = 500000; /\* setPWM cycle\*/

if(0 == channel)

LPC\_PWM1->MR3 = offset; // Motor

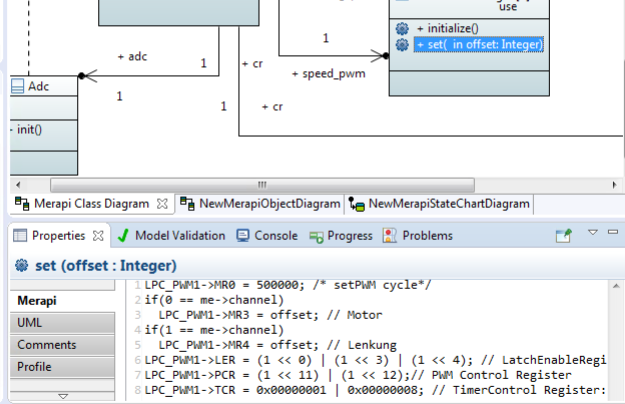
if(1 == channel)

LPC\_PWM1->MR4 = offset; // Lenkung

LPC\_PWM1->LER = (1 << 0) | (1 << 3) | (1 << 4); // LatchEnableRegister

LPC\_PWM1->PCR = (1 << 11) | (1 << 12);// PWM Control Register

LPC\_PWM1->TCR = 0x00000001 | 0x00000008; // TimerControl Register: Counter enablePWM enable



* In der Klasse Adc wird in der init-Methode den portclock(pclk) initialisiert, den Strom eingeschaltet und die ADC Pins als Input konfiguriert.

intADC\_CLK = 1000000; // 1MHz

LPC\_SC->PCONP |= (1 << 12); // Power einschalten

/\* ADC Pins als Input konfigurieren \*/

LPC\_PINCON->PINSEL1 |= (1U<<14); /\* ADC0 \*/

LPC\_PINCON->PINSEL1 |= (1U<<16); /\* ADC1 \*/

LPC\_PINCON->PINSEL1 |= (1U<<18); /\* ADC2 \*/

LPC\_PINCON->PINSEL1 |= (1U<<20); /\* ADC3 \*/

LPC\_PINCON->PINSEL3 |= (3U<<28); /\* ADC4 \*/

LPC\_PINCON->PINSEL3 |= (3U<<30); /\* ADC5 \*/

LPC\_PINCON->PINSEL0 |= (2U<<6); /\* ADC6 \*/

LPC\_PINCON->PINSEL0 |= (2U<<4); /\* ADC7 \*/

for (i = 0; i < 5000000; i++); // Delay..

/\* By default, the PCLKSELxvalue is zero, thus, the PCLK for all the peripherals is 1/4 of the SystemCoreClock. Bit 24~25 is for ADC \*/

pclkdiv= (LPC\_SC->PCLKSEL0 >> 24) & 0x03;

switch(pclkdiv) {

case0x00:

default:

pclk= SystemCoreClock/ 4;

break;

case0x01:

pclk= SystemCoreClock;

break;

case0x02:

pclk= SystemCoreClock/ 2;

break;

case0x03:

pclk= SystemCoreClock/ 8;

break;

}

LPC\_ADC->ADCR = (0x01 << 0) | /\* SEL=1,select channel 0~7 on ADC0 \*/

((pclk/ ADC\_CLK -1) << 8) | (0 << 16) |/\* BURST = 0, softwarecontrolled\*/

(0 << 17) | /\* CLKS = 0, 11 clocks/10 bits \*/

(1 << 21) | /\* PDN = 1, normal operation\*/

(0 << 24) | /\* START = 0 A/D conversion stops \*/

(0 << 27); /\* EDGE = 0 falling,triggerA/D \*/

/\* Start bits need to be zero before BURST mode can be set. \*/

if (LPC\_ADC->ADCR & (0x7 << 24)) {

LPC\_ADC->ADCR &= ~(0x7 << 24);

}

LPC\_ADC->ADCR &= ~0xFF;

/\* Read all channels, 0 through 7. \*/

LPC\_ADC->ADCR |= 0xFF;

LPC\_ADC->ADCR |= (0x1 << 16); /\* Set burst mode and start A/D convert \*/

Die Implementierung in querySensors

me->ADCValue[0] = (LPC\_ADC->ADDR0) >> 4) & 0xFFF;

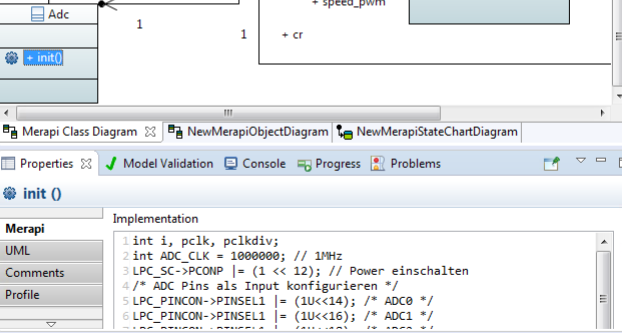
me->ADCValue[1] = (LPC\_ADC->ADDR1) >> 4) & 0xFFF;

me->ADCValue[2] = (LPC\_ADC->ADDR2) >> 4) & 0xFFF;

me->ADCValue[3] = (LPC\_ADC->ADDR3) >> 4) & 0xFFF;

me->ADCValue[4] = (LPC\_ADC->ADDR5) >> 4) & 0xFFF;

* Das untere Bild ist eine Illustration davon:



## Object Diagram dient zur Struktur des Programms

Das Objektdiagramm ist ein Strukturdiagramm, denn es zeigt eine bestimmte Sicht auf die Struktur des modellierten Systems, wobei jedes Objekt für eine Klasse steht. Die Darstellung umfasst dabei typischerweise Ausprägungsspezifikationen von Klassen und Assoziationen.

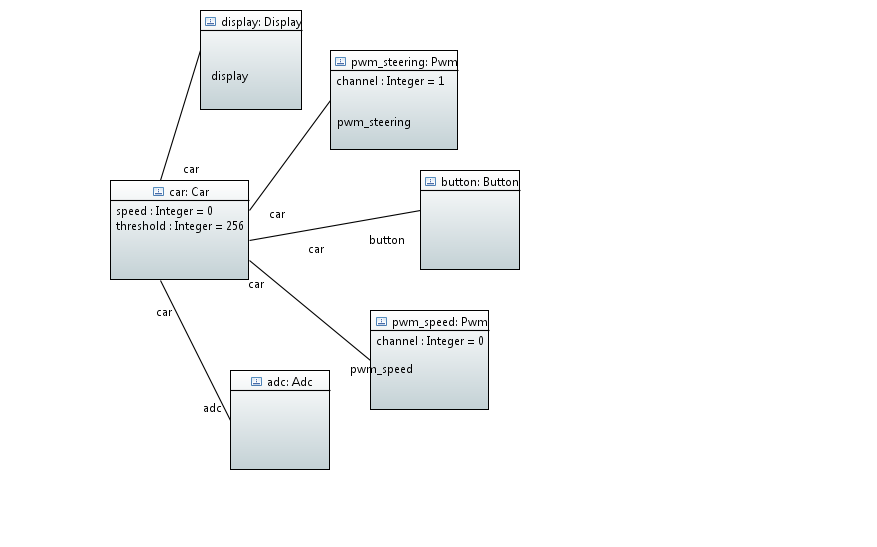
In diesem Diagramm spielt "Car" die main-Methode Rolle. Andere Methoden gehören zu Car-Methode. "Display" kontrollieren die Auszeichen des Bildschirms.

"pwm-steering" kontrolliert die Lenkung des Autos.

"Button"

"pwm-speed" kontrolliert die Geschwindigkeit des Autos. Wir müssen unbedingt eine passende Geschwindigkeit auswählen. Wenn die Geschwindigkeit zu schnell ist, werden sich die Räder des Autos rückwärtsdrehren.

"adc" Kontrolliert die Lenkung wenn das Auto gefahren wird. Sensoren kann das Licht messen. Durch A/D Wandler, kann man das Signal bekommen und bearbeiten.

****

## State Diagram steht das Verfahren von dem gesamten Programm

* Im „init“ wird die zu durchführenden Schritte als Methoden deklariert(siehe untere Bild)

int i;

SystemInit();

Button\_init(me->);

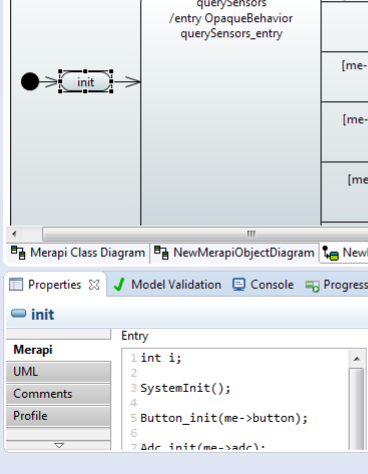
ADC\_init(me->adc）;

Pwm\_initialize(me->speed\_pwm);

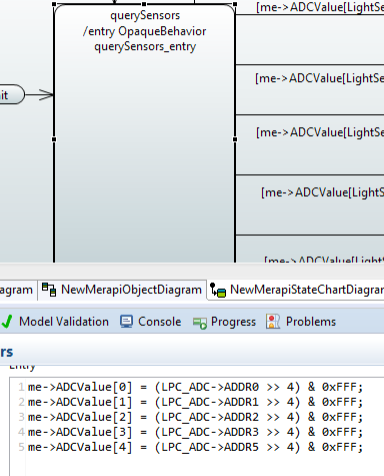
Pwm\_set(me->speed\_pwm, 30000);

Button\_wait(me->button);

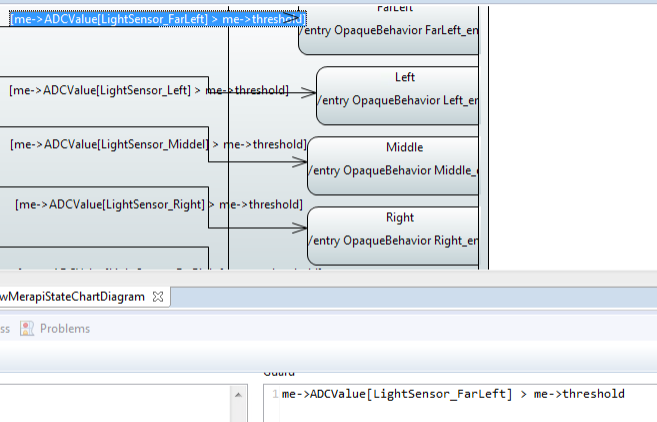
Pwm\_set(me->speed\_pwm, 27500);



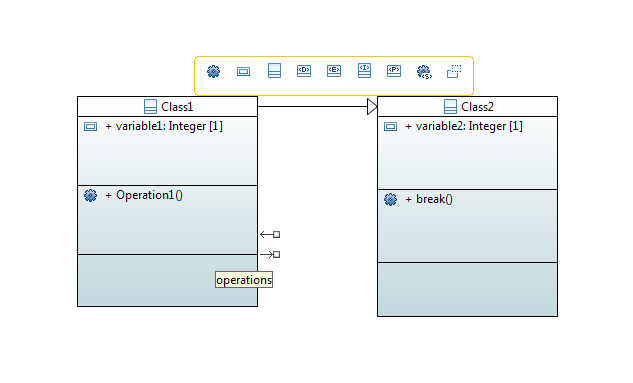
* Im „querySensors“ werden die Werte der ADCValue deklariert und initialisiert. Mehrere Vorstellung kann man sich mit dem Unteren Bild vorstellen



* Letztes werden die entsprechenden guard[me->ADCValue[LightSensor\_FarLeft]> me->threshold] zu den dazugehörigen Steuerungen eingerichtet (zum Beispiel FarLeft).Das ganze sieht man wieder auf dem untere Bild



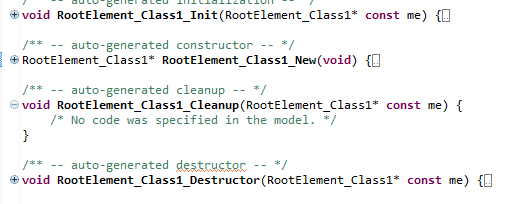
* Das geschriebene Programm in Merapi lässt sich zusammen in Keil darstellen und mit-Kompilieren. Hier Nochmal die komprimierteZip-Datei davon:



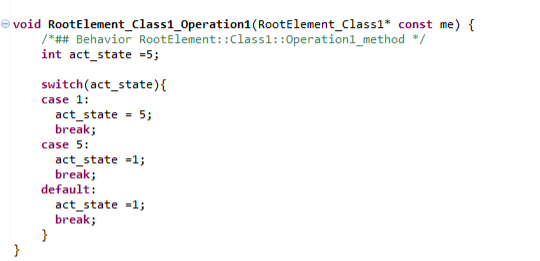
# Verhältnis von Klassen

Ziel der Aufgabe war sich den Verhältnis von Class1 und Class2 anschauen

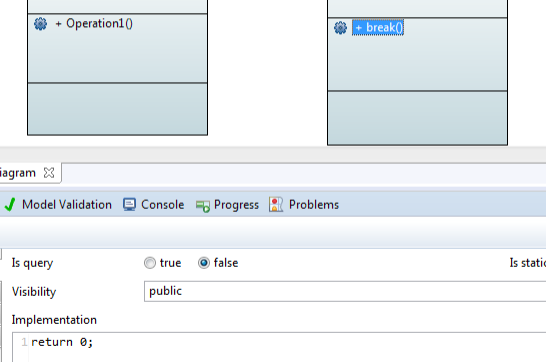
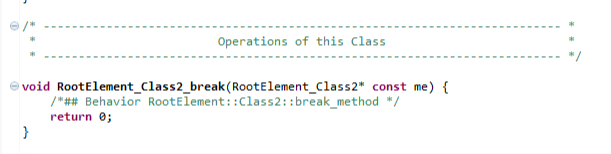
* Wenn wir im UML Klassendiagramm eine Klasse anlegen, erzeugt sich diese automatisch unter den Papyrus Projekt
* Auf diesem Klassendiagramm stehen Class1 und Class2 die mit eine Aggregation zueinander verkoppelt sind
* Durch dem Symbol wird angedeutet, dass die Methoden von Class2 in Class1 sind
* Es wird die Funktionen "void RootElement\_Class\_Inti()" , "RootElement\_Class\* Rootelement\_Class\_New()" , "void RootElement\_Class\_Cleanup()" und "void RootElement\_Class\_Destructor()" automatisch erzeugt. Alle Klassen müssen diese vier Funktionen haben.(siehe unterem Bild)



* In Class1.c wird die in Class1.h definierte Methode(Operation1) implementiert(siehe unterem Bild).Sonst wird in Class1.h und Class2.h die variable1 und 2 jeweils definiert, zusätzlich wird die variable2 und Methode break von Class2 in Class1.h definiert.
* Die in Class1.h und Class2.h definierten Methoden werden beziehungsweise in Class1.c und Class2.c implementiert. Da meistens in h-Files Definition stehen und in c-Files Implementierung.



* Code schreibt man unter Implementation beim Class diagram und taucht automatisch in der entsprechenden c-Datei. Zum Beispiel die Methode break von Class2.Illustration davon sieht man auf den folgenden Bilder:

# XTEND Programmiersprache

In diese Übung sollten wir den vorliegenden Code korrigieren. Code testen läuft in JUnit, die eine Einheit des Test-Frameworks für Java-Programme. In JUnit suchen wir, wo die Fehler sind. Danach korrigieren wir diese. Wir sollten zwei Teile Machen. Eine ist über Parameter und das andere über Operation.

* Folgend sind einige JUnit-Test,die wir gelöst haben und die dazugehörigen Lösungen, die wir herstellt haben.

. blau sind Test

. rot sind Lösungen

## TestOperation

@Test def testOneParameterOperation() {

val operation = createOperation => [

**name = "oneParameterOperation"**

......

val code = (new Uml2C).generateCode(operation)

**Assert.assertEquals("void oneParameterOperation(TestClass\* const me, uint32 param1);", code)**

}

if(**operation.name == "oneParameterOperation"**){

){

**return '''void (TestClass\* const me, uint32 param1);'''**

}

@Test def testTwoParameterOperation() {

val operation = createOperation => [

**name = "twoParameterOperation"**

......

val code = (new Uml2C).generateCode(operation)

**Assert.assertEquals("void twoParameterOperation(TestClass\* const me, uint32 param1, uint8 param2);", code)**

}

else if(operation.name == "twoParameterOperation" ){

return '''void twoParameterOperation(TestClass\* const me, uint32 param1, uint8 param2);'''

}

@Test def testReturningOperation() {

val operation = createOperation => [

name = "returningOperation"

......

val code = (new Uml2C).generateCode(operation)

Assert.assertEquals("uint32 returningOperation(TestClass\* const me);", code)

}

else if(**operation.name == "returningOperation"** ){

**return '''uint32 returningOperation(TestClass\* const me);'''**

}

@Test def testImplementedOperation() {

val operation = createOperation => [

**name = "implementedOperation"**

......

Assert.assertEquals(

'''

**void implementedOperation(TestClass\* const me) {**

**/\* hier kaennte Ihre Werbung stehen \*/**

**}**

**'''.toString, code)**

}

else if(**operation.name == "implementedOperation"** ){

**return '''**

**void implementedOperation(TestClass\* const me) {**

**/\* hier kaennte Ihre Werbung stehen \*/**

}

'''

}

@Test def testPrivateOperation() {

val operation = createOperation => [

**name = "privateOperation"**

......

**Assert.assertEquals("static void privateOperation(TestClass\* const me);", code)**

}

else if(**operation.name == "privateOperation"** ){

**return '''static void privateOperation(TestClass\* const me);'''**

}

@Test def testStaticOperation() {

val operation = createOperation => [

**name = "staticOperation"**

......

**Assert.assertEquals("void staticOperation(uint32 param1);", code)**

}

else if(**operation.name == "staticOperation"** ){

**return '''void staticOperation(uint32 param1);'''**

}

@Test def testQueryOperation() {

val operation = createOperation => [

**name = "queryOperation"**

......

**Assert.assertEquals("void queryOperation(const TestClass\* const me);", code)**

}

else if

(**operation.name == "queryOperation"** ){

return '''void queryOperation(const TestClass\* const me);'''

}

## TestParameter

@Test def testPrimitiveInOutParameter() {

val parameter = createParameter => [

name = "primitiveInOutParameter"

**direction = ParameterDirectionKind.INOUT\_LITERAL**

type = createPrimitiveType => [name = "uint32"]

]

val code = (new Uml2C).generateCode(parameter)

**Assert.assertEquals("uint32\* primitiveInOutParameter", code)**

}

if(**ParameterDirectionKind.INOUT\_LITERAL** == umlParameter.direction &&

umlParameter.type instanceof PrimitiveType

){

**return '''«generateType(umlParameter.type)»\* «umlParameter.name»'''**

}

val parameter = createParameter => [

name = "primitiveReturnParameter"

direction = **ParameterDirectionKind.RETURN\_LITERAL**

type = createPrimitiveType => [name = "uint32"]

]

val code = (new Uml2C).generateCode(parameter)

**Assert.assertEquals("uint32", code)**

}

else if( **ParameterDirectionKind.RETURN\_LITERAL** == umlParameter.direction){

**return '''«generateType(umlParameter.type)»'''**

}

@Test def testComplexOutParameter() {

val parameter = createParameter => [

name = "complexOutParameter"

direction = **ParameterDirectionKind.OUT\_LITERAL**

type = createClass => [name = "ComplexType"]

]

val code = (new Uml2C).generateCode(parameter)

**Assert.assertEquals("ComplexType\*\* complexOutParameter", code)**

}

else if(**ParameterDirectionKind.OUT\_LITERAL** == umlParameter.direction){

**return '''«generateType(umlParameter.type)»\* «umlParameter.name»'''**

}

@Test def testReadParameter() {

val parameter = createParameter => [

name = "readParameter"

effect = **ParameterEffectKind.READ\_LITERAL**

type = createClass => [name = "ComplexType"]

]

val code = (new Uml2C).generateCode(parameter)

**Assert.assertEquals("ComplexType\* const readParameter", code)**

}

else if(**ParameterEffectKind.READ\_LITERAL** == umlParameter.effect ){

**return '''«generateType(umlParameter.type)» const «umlParameter.name»'''**

}

# Aktivitäts- Diagramm

Diagramm anlegen geht über: Papyrus projekt-neu-Papyrus modell-name eingeben-create new-activity Diagram.Ergebnis wird auf dem folgenden Bild angezeigt:

