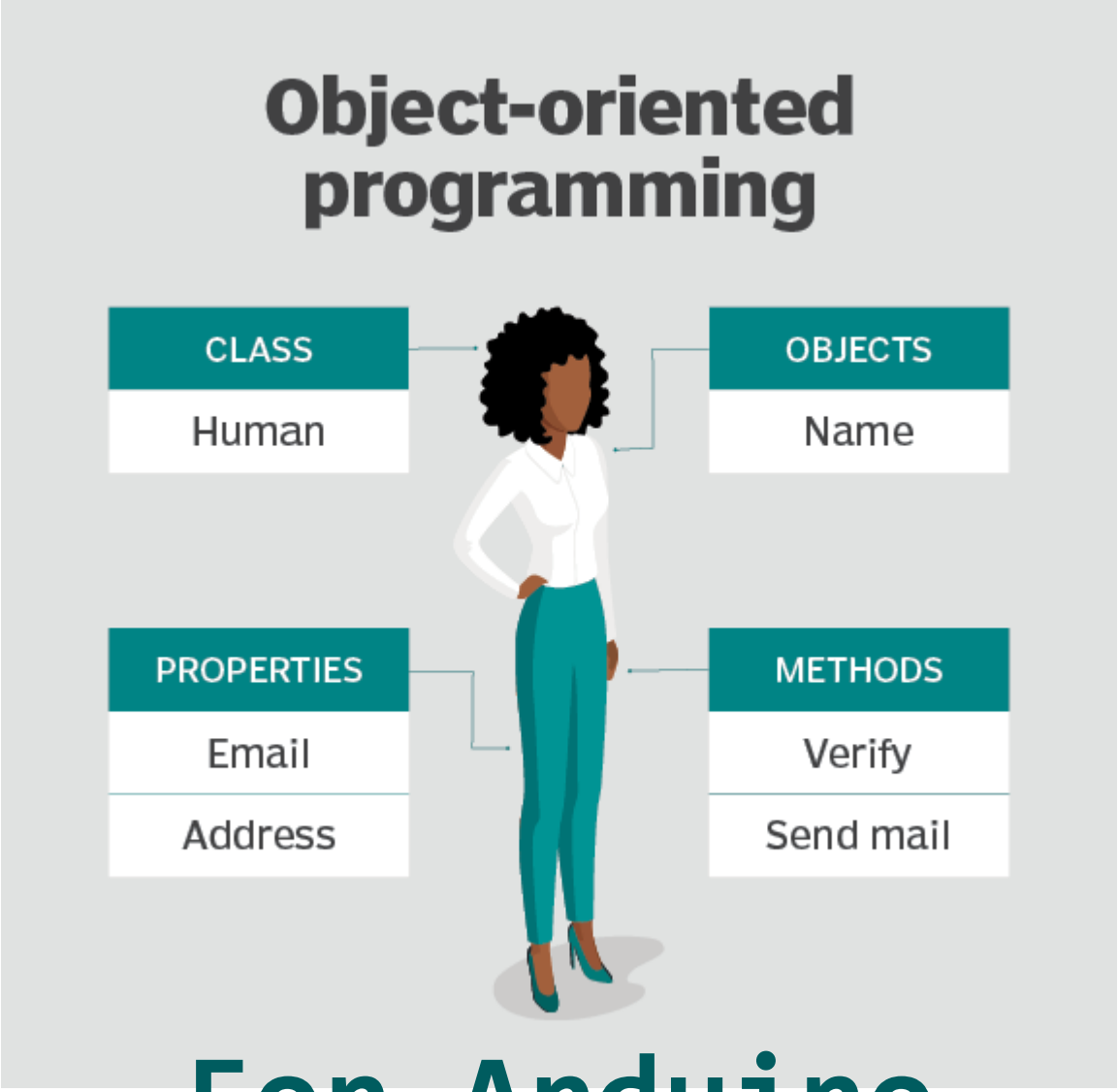

Object-oriented programming
CLASS
Human
OBJECTS
Name
PROPERTIES
Email
Address
METHODS
Verify
Send mail

For Arduino

Inhoud

De basisprincipes van OOP: 5
Verschillen en overeenkomsten tussen functies en OOP 6
Abstractie en OOP in de Arduino omgeving 8
Elektronisch schema voor deze cursus: 9
Eerste OOP programma – led blinker 10
Led blinker met OOP principes – alles in 1 file 12
Led blinker met OOP principes met header en cpp file 14
*.ino file 14
Led.h file 15
Led.cpp file 16
Led blinker met OOP principes met header en cpp file in Arduino library 17
Pushbutton versie 1 18
PushButton.ino 18
PushButton.h 18
PushButton.cpp 19
Pushbutton versie 2 met PULL_UP 20
PushButton.ino 21
PushButton.h 21
PushButton.cpp 22
Pushbutton versie 3 met Debounce 24
PushButton.ino 24
PushButton.h 24
PushButton.cpp 25
Pushbutton en leds samen in 1 programma 28
OOP_les3.ino 28
Led Blinker met overerving 30
LedBlinker STAP 1 Overerving 31
Led_blinker.ino 31
Led.h 32
Led.cpp 32
LedBlinker.h 33
LedBlinker.cpp 34
LedBlinker STAP 2 Toggle 35
Led_blinker.ino 35

Led.h 36
Led.cpp 36
LedBlinker.h 38
LedBlinker.cpp 38
LedBlinker STAP 3 Blinkerfrequentie 40
Led_blinker.ino 40
Led.h 40
Led.cpp 41
LedBlinker.h 42
LedBlinker.cpp 43
LedBlinker STAP 4 Getter en Setter 45
Led_blinker.ino 45
Led.h 46
Led.cpp 46
LedBlinker.h 48
LedBlinker.cpp 48
Extra Uitdagingen LedBlinker: 50
Traffic Light 51
Schema 51
UML voorstelling van het afgewerkte project 52
Traffic Light step 1 Overerving 53
TrafficLight.h 53
TrafficLight.cpp 54
Traffic_Light STEP1.ino 55
Traffic Light step 2 – StateMachine 57
TrafficLight.h 62
TrafficLight.cpp 63
TrafficLight STEP 2 STATE Machine.ino 65
Traffic Light step 3 – Potmeter 67
Potentiometer.h 67
Potentiometer.cpp 67
Led.h 68
Led.cpp 68
TrafficLight.h 70
Trafficlight.cpp 71

Final program.ino 74

Object georiënteerd programmeren & Arduino

Object-Oriented Programming (OOP) is een programmeerparadigma dat is gebaseerd op het concept van "objecten", die data en de bijbehorende functionaliteit combineren.

De basisprincipes van OOP:

Abstraction (Abstractie):
Abstractie is het proces waarbij je ervoor gaat zorgen dat je je bij het schrijven van een programma enkel nog moet bezig houden met de essentie en dat je alle complexe code verstopt in een klasse.

Encapsulation (Encapsulatie):
Encapsulatie houdt in dat je de klasse die je schrijft zoveel mogelijk beschermd voor de buitenwereld door de buitenwereld enkel toegang te geven tot de klasse via goed gedefinieerde methoden. We verwezenlijken dit voornamelijk door zoveel als mogelijk van de eigenschappen ‘private’ te maken.

Inheritance (Overerving):
Overerving is een mechanisme waarbij een nieuwe klasse (child class) de eigenschappen en methoden erft van een bestaande klasse (parent class). Het stelt programmeurs in staat om codehergebruik te bevorderen, omdat de child class automatisch alle kenmerken van de parent class overneemt en ook zijn eigen specifieke kenmerken kan toevoegen.

Polymorphism (Polymorfisme):
Polymorfisme betekent letterlijk "veelvormigheid" en verwijst naar het vermogen van objecten om verschillende vormen aan te nemen, afhankelijk van de context waarin ze worden gebruikt. Dit kan zich uiten in methoden die verschillend gedrag vertonen afhankelijk van de parameters of in klassen die dezelfde methode op verschillende manieren implementeren. Polymorfisme wordt niet toegepast in deze cursus.

Verschillen en overeenkomsten tussen functies en OOP

Er zijn 3 types functies:

Niets in en niets uit

```
puls();
```

```
void puls(void)
{
digitalWrite (13,HIGH);
delay(100);
digitalWrite (13,LOW);
}
```

Iets in en niets uit

```
pulsPin(13);
```

```
void pulsPin(int x)
{
digitalWrite (x,HIGH);
delay(100);
digitalWrite (x,LOW);
}
```

iets in en iets uit

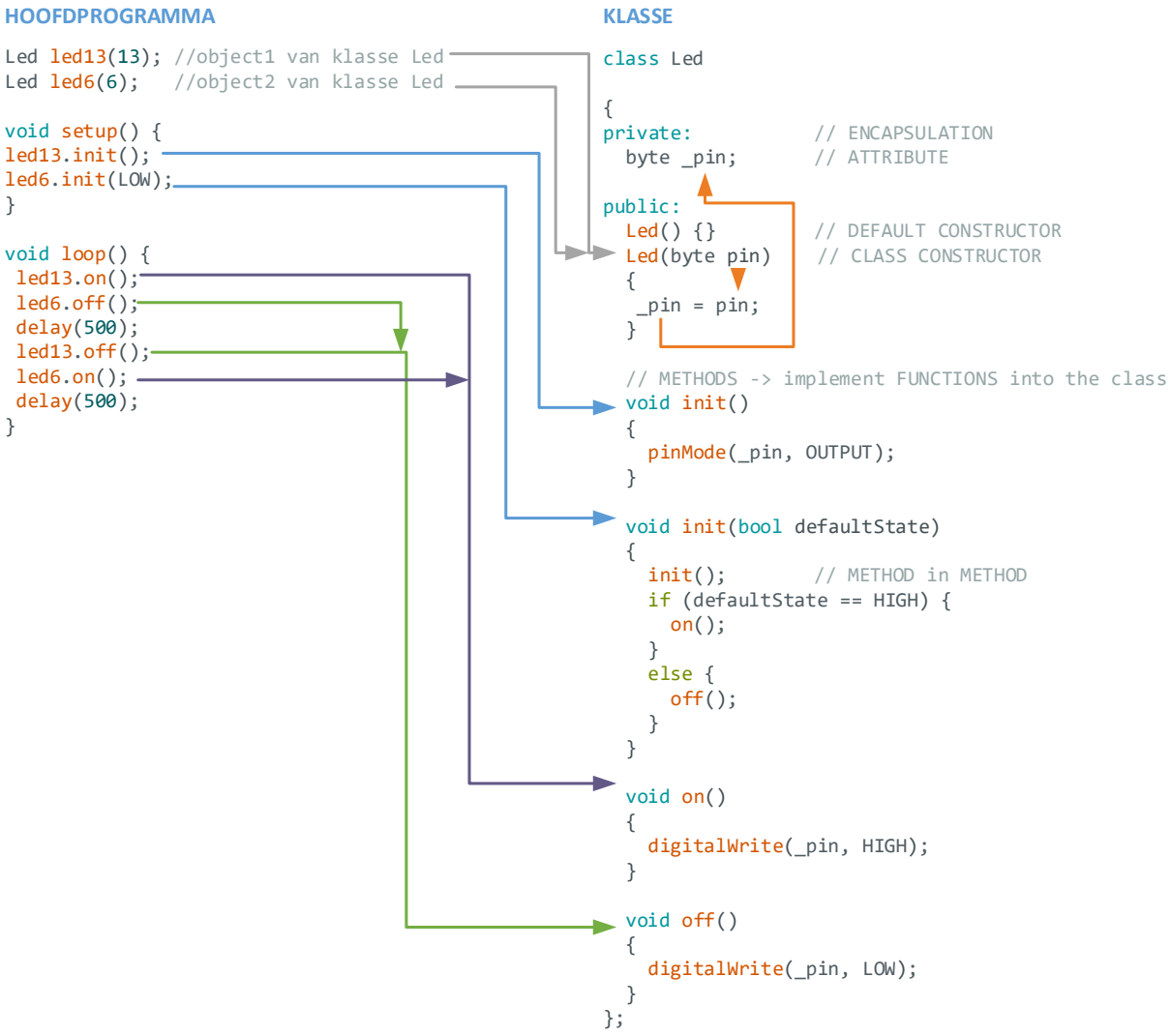
```
int result = som(11,6);
```

```
int som(int y, int z)
{
int optel = y + z;
return optel;
}
```

Functies – die wij hier als voorloper van OOP zien – hebben een aantal eigenschappen en beperkingen

- Een functie kan worden aangeroepen maar er dient dan wel **meteen het juiste aantal attributen** te worden meegegeven (x, y, z in bovenstaande voorbeelden).
- Een functie **kan slechts 1 resultaat teruggeven** (via return)
- Een functie dient afgewerkt te zijn voordat die de volgende keer kan worden aangeroepen. Anders worden de variabelen overschreven.
- Doel van gebruik van functies = abstractie
- Door lokale variabelen te gebruiken beschermen we de functie = encapsulatie

Gebruik van OOP met klassen en objecten geeft ons meer mogelijkheden:



- Wanneer een klasse meerdere keren wordt aangeroepen - door er meerdere objecten van te maken – zullen de variabelen van die klasse gekopieerd worden in het werkgeheugen (mirror) – zo kunnen de objecten naast elkaar gebruikt worden zonder mekaar te verstoren. .
- Een klasse kan een verzameling van functies of methoden (andere naam voor functie) bevatten
- Een methode kan met een verschillend aantal **attributen** worden aangeroepen = overload constructor
- Van een klasse kunnen er verschillende resultaten worden teruggevraagd
- Doel van gebruik van klassen = abstractie
- Door lokale variabelen te gebruiken beschermen we de klasse = encapsulatie
- Klassen kunnen worden hergebruikt in andere klassen = enheritance
- Objecten kunnen verschillende vormen aannemen = polymorfism

Abstractie en OOP in de Arduino omgeving

De Arduino omgeving maakt gebruik van C++ en van oneindig veel bibliotheken die als voornaamste doel hebben om heel complexe code en functies zo ABSTRACT te maken dat gebruikers de functionaliteit ervan – zonder diepgaande kennis van hoe alles er achter juist werkt – toch kunnen gebruiken.

"ABSTRAHEREN" in de context van Object-Oriented Programming (OOP) betekent simpelweg het verbergen van ingewikkelde details en alleen laten zien wat belangrijk is voor het gebruik van iets.

Stel dat je de ogen van dit spookje wil programmeren met leerlingen van een 6e leerjaar.

Je kan dit doen als volgt:

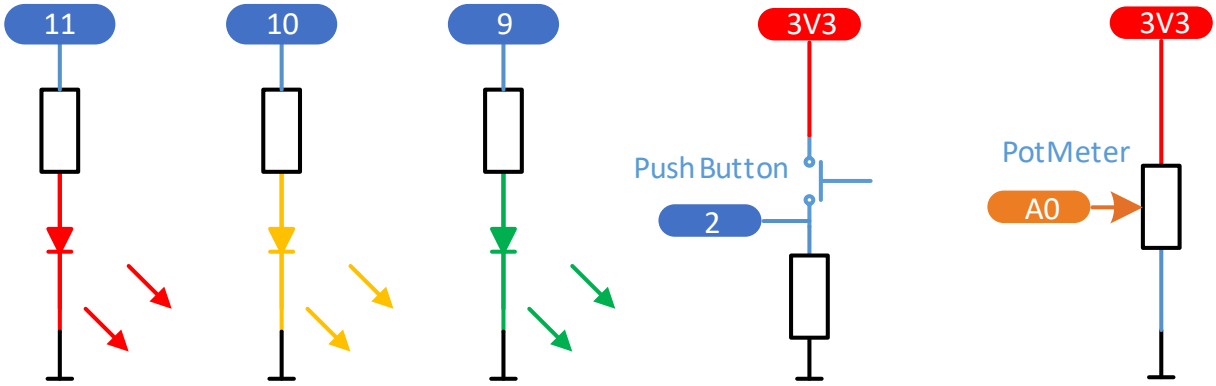
```
void loop() {
        digitalWrite(6, HIGH);
        digitalWrite(7, LOW);
        delay(500);
        digitalWrite(6, HIGH);
        digitalWrite(7, LOW);
        delay(500); }
```

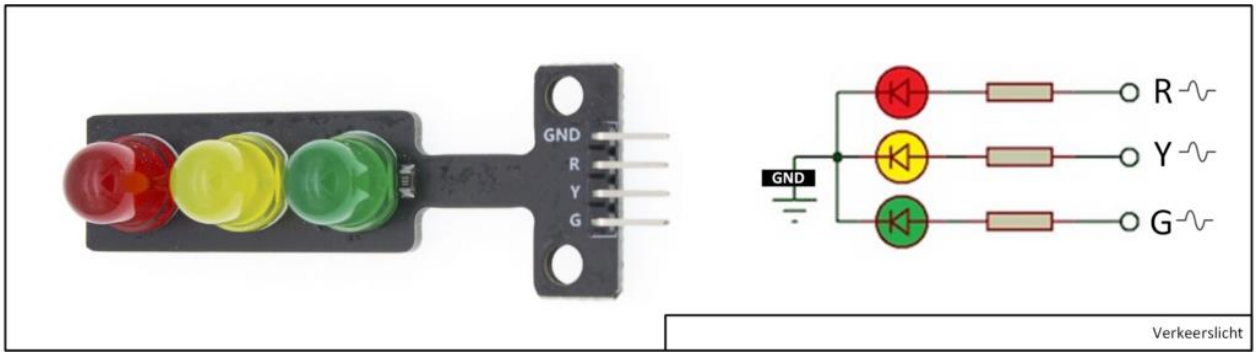
Of met een extra level van abstractie:

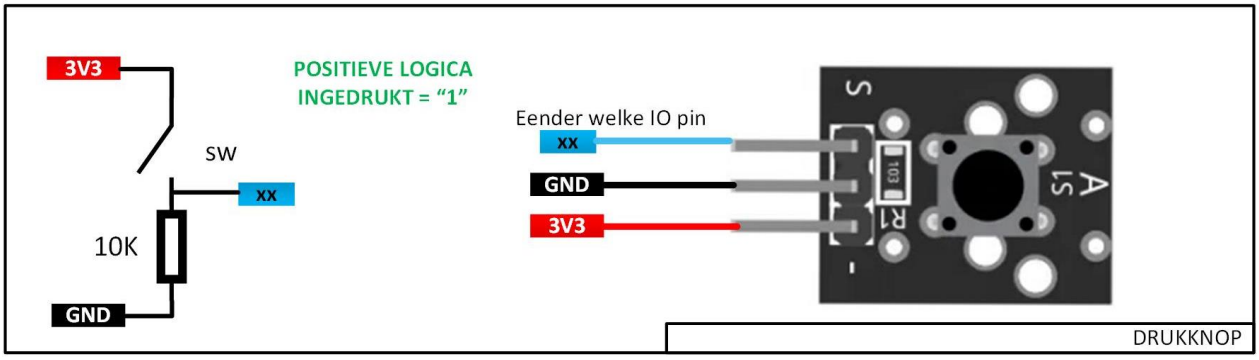
```
void loop() {
        oogLinks.aan();
        oogRechts.uit();
        wacht(500);
        oogLinks.uit();
        oogRechts.aan();
        wacht(500); }
```

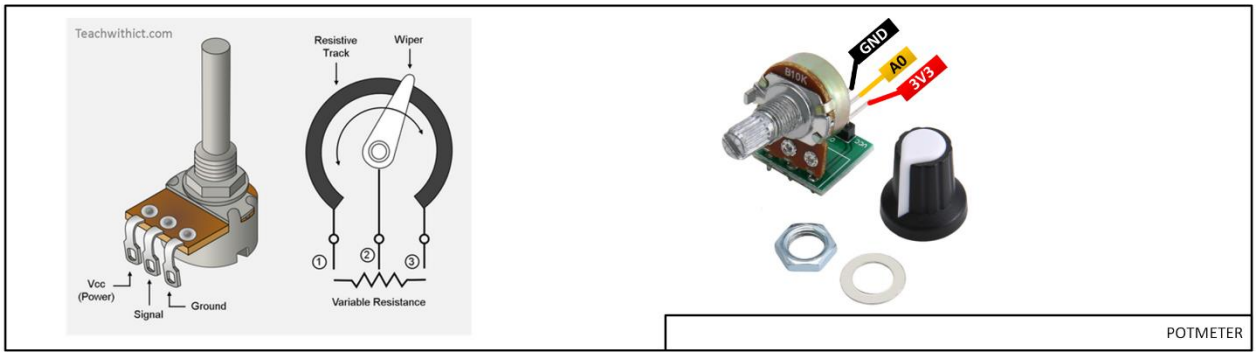
De oorsponkelijke bibliotheken die bij Arduino geleverd werden, zijn voornamelijk geschreven door gebruik te maken van functies, maar naarmate meer en meer Arduinogebruikers ook kennis kregen van OOP werden ook de nieuwere Arduino bibliotheken meer en meer volgens de OOP principes geprogrammeerd.
OOP code leest op het eerste zicht wat moeilijker, maar is vooral veel makkelijker te 'onderhouden'.

Elektronisch schema voor deze cursus:









Eerste OOP programma – led blinker

We starten ons eerste OOP Arduino programma bij de led blinker die we allemaal kennen:

```
#define LED_PIN 13

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(500);
  digitalWrite(LED_PIN, LOW);
  delay(500);
}
```

Op zich is dit al een redelijk abstract programma – zeker als je het vergelijkt met hoe de code er zou uit zien als we geen gebruik mogen maken van de Arduino bibliotheken:

```
#include <avr/io.h>
#include <util/delay.h>

int main(void) {
    // Zet de pin als output
    DDRB |= (1 << DDB5); // DDB5 komt overeen met pin 13
    while (1) {
        PORTB |= (1 << PORTB5); // Led aan - PORTB5 komt overeen met pin 13
        _delay_ms(500);
        PORTB &= ~(1 << PORTB5); // Led uit - PORTB5 komt overeen met pin 13
        _delay_ms(500);
}}
```

We hebben in dit geval zelf de code geschreven om pins hoog of laag te maken. We hebben daarvoor de juiste registernamen van deze Arduino processor moeten opzoeken in de desbetreffende datasheets. Er is hier dus veel meer kennis nodig om een identiek programma te schrijven. We spreken hier dus van **minder abstractie.**

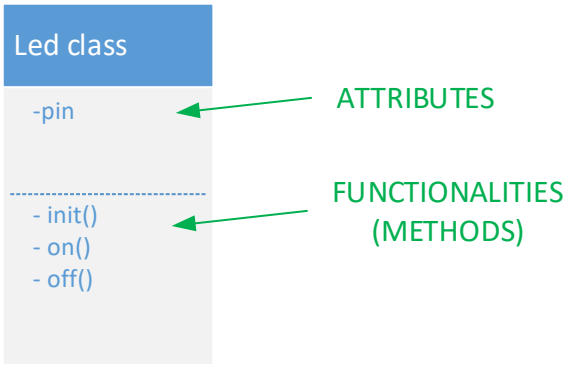
En toch kunnen we ons eerste blinker programma nog abstracter schrijven:
De eerste stap van een OOP programma is steeds 'nadenken over wat je wenst te doen'

Wat willen we doen?
We willen een led laten knipperen

Welke **ATTRIBUTES** hebben we nodig? (**Wat willen we kunnen meegeven / terugkrijgen**?)
Pinnummer van de led

Welke **FUNCTIONALITIES** willen we hebben (**Wat moet de class DOEN**?)
init() -> pin als output zetten
on() -> pin hoog maken
off() -> pin laag maken

Dit wordt grafisch meestal voorgesteld door een **UML (Unified Modeling Language)** tekening



Led blinker met OOP principes – alles in 1 file

```
#define LED_PIN 13

class Led       // naam beginnen met Uppercase, daarna Lower case
                // DOEL: functionaliteiten meer ABSTRACT schrijven (ABSTRACTIE)
{
private:          // alles onder private: is enkel beschikbaar binnen de class
                  // ENCAPSULATION: private variabelen en functies
                  // die niet beschikbaar zijn buiten de class
  byte _pin;      // ATTRIBUTE -> private var om pin nr in te lezen in de class

public:     // alles onder public is beschikbaar van buiten de class
  Led() {}    // do not use   // DEFAULT CONSTRUCTOR heeft geen parameters
              // nodig om compilatie errors te voorkomen bij ENHERITANCE

  Led(byte pin)    // CLASS CONSTRUCTOR = function/method in een class
                   // (byte pin = ATTRIBUTE)
                   // Led naam moet exact dezelfde zijn als de naam van de class
                   // Deze constructor wordt aangeroepen wanneer je
                   // een OBJECT of INSTANCE van THE CLASS aanmaakt

                   // FUNCTION OVERLOADING: >1 CLASS CONSTRUCTORS met
                   //verschillend aantal ATTRIBUTES
  {
    //this->pin = pin;   // pinnummer vanuit hoofdprogramma wordt gekopieerd naar
                         //private pin variabele (heel verwarrend)
     _pin = pin;         //_pin = pin;  is veel duidelijker als de private var
                         //ook _pin genoemd wordt (maar wordt minder gebruikt)
  }

  // METHODS -> implement FUNCTIONS into the class

  void init()
  {
    pinMode(_pin, OUTPUT);
  }

  void init(bool defaultState)
  {
    init();                     // vb van gebruik van METHODS in andere METHOD
    if (defaultState == HIGH) {
      on();
    }
    else {
```

```
      off();
    }
  }

  void on()
  {
    digitalWrite(_pin, HIGH);
  }

  void off()
  {
    digitalWrite(_pin, LOW);
  }
};    // class altijd eindigen met ;


// start van het eigenlijke programma
Led led13(LED_PIN);  // aangeraden methode
                  // ook mogelijk via DYNAMIC ALLOCATION met new keyword
                     // Led *led13 = new Led(LED_PIN);

void setup() {
  led13.init(LOW);
}

void loop() {
  led13.on();       // meer context - meer ABSTRACTIE
  delay(500);
  led13.off();
  delay(500);
}
```

Led blinker met OOP principes met header en cpp file

OOP_LES1_LED.ino | Led.h | Led.cpp

```
1   #define LED_PIN 13
2   #include "Led.h"   // icluden van de .h file
3
4
5   Led led13(LED_PIN);  // aangeraden methode
6                        // ook mogelijk via DYNAMIC ALLOCATION met new keyword
7
8   void setup() {
9     led13.init(LOW);
10  }
11
12  void loop() {
13    led13.on();       // meer context - meer ABSTRACTIE
14    delay(500);
15    led13.off();
16    delay(500);
17  }
18
```

*.ino file

```
#define LED_PIN 13
#include "Led.h"   // includen van de .h file


Led led13(LED_PIN);  // aangeraden methode
                     // ook mogelijk via DYNAMIC ALLOCATION met new keyword

void setup() {
  led13.init(LOW);
}

void loop() {
  led13.on();       // meer context - meer ABSTRACTIE
  delay(500);
  led13.off();
  delay(500);
}
```

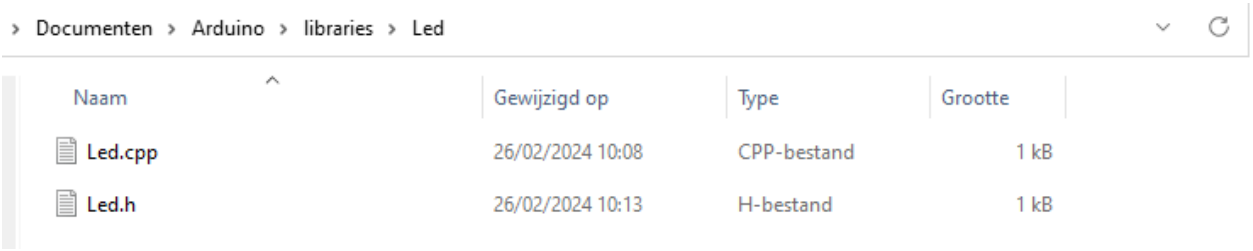
Led.h file

```
// HEADER GUARDS (if Not Defined)
// GUARD zorgt ervoor dat deze code slechts 1x gedupliceerd kan worden
#ifndef LED_H
#define LED_H

#include <Arduino.h>  // moeten we hier handmatig includen

// in*.h -> DECLARATION & INTERFACE
class Led
{
private:

  byte _pin;

public:
  // CONSTRUCTORS
  Led() {}    // We leave the default constructor in the header file
  Led(byte pin);

  // METHODS
  void init(); // init the pin as OUTPUT - call from SETUP
  void init(bool defaultState); // init the pin as OUTPUT (LOW / HIGH) - call
from SETUP
  void on(); // Make output pin HIGH
  void off(); // Make output pin LOW
};

#endif
```

Led.cpp file

```
#include "Led.h"    // We also need to include Led.h from within the *.cpp file

// in*.cpp -> IMPLEMENTATION OR LOGIC of the CLASS

// CONSTRUCTOR
Led::Led(byte pin)     // implement Led constructor from Led class
{
     _pin = pin;
}

// METHODS
void Led::init()    // implement init from Led class
{
  pinMode(_pin, OUTPUT);
}

void Led::init(bool defaultState)
{
  init();
  if (defaultState == HIGH) {
    on();
  }
  else {
    off();
  }
}

void Led::on()
{
  digitalWrite(_pin, HIGH);
}

void Led::off()
{
  digitalWrite(_pin, LOW);
}
```

Led blinker met OOP principes met header en cpp file in Arduino library

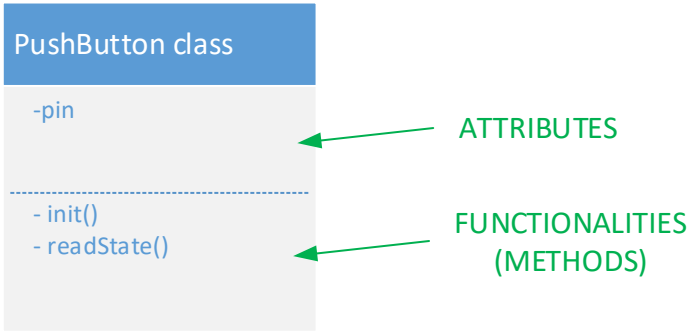



```
#define LED_PIN 13
#include <Led.h>   // includen van de .h file – nu onder Arduino\libraries\led


Led led13(LED_PIN);


void setup() {
  led13.init(LOW);
}

void loop() {
  led13.on();       // meer context - meer ABSTRACTIE
  delay(500);
  led13.off();
  delay(500);
}
```

Uitdaging: pas bovenstaande code aan zodat de 3 leds van het verkeerslicht kunnen gestuurd worden.

Pushbutton versie 1

Opgave: In versie 1 van deze code willen we een klasse die de initialisatie van de pin als INPUT verzorgt en die ons een waarde teruggeeft die aangeeft of de drukknop wel of niet ingedrukt is.



PushButton.ino

```
#include "PushButton.h"

PushButton knop(2);  // object van class PushButton

void setup()
{
Serial.begin(9600);
knop.init();
}

void loop()
{
Serial.println(knop.readState());
delay(100);
}
```

PushButton.h

```
// HEADER GUARDS (if Not Defined)
// GUARD zorgt ervoor dat deze code slechts 1x gedupliceerd kan worden
#ifndef PUSH_BUTTON_H
#define PUSH_BUTTON_H

#include <Arduino.h>

// in*.h -> DECLARATION & INTERFACE
class PushButton
{
```

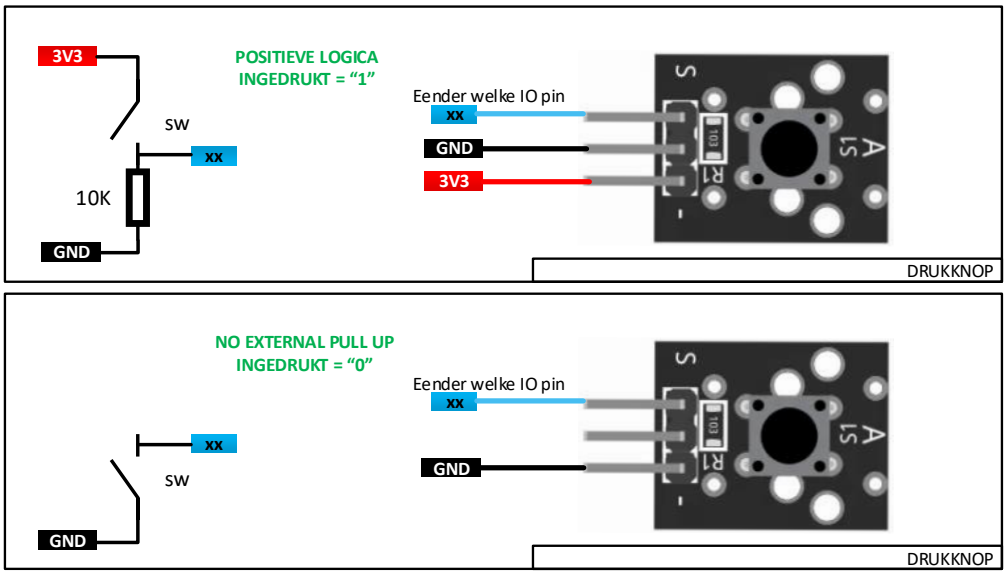
```
private:
  byte _pin;

public:
  // CONSTRUCTORS
  PushButton() {}
  PushButton(byte pin);

  // METHODS
  void init();
  bool readState();
};

#endif
```

PushButton.cpp

```
#include "PushButton.h"

// in*.cpp -> IMPLEMENTATION OR LOGIC of the CLASS

// CONSTRUCTOR
PushButton::PushButton(byte pin)
{
     _pin = pin;
}

// METHODS
void PushButton::init()
{
 pinMode(_pin, INPUT);
}

bool PushButton::readState()
{
  if (digitalRead(_pin)) return true;  // return = geef terug
  else return false;
}
```

Pushbutton versie 2 met PULL_UP

Drukknoppen kunnen op verschillende manieren worden aangesloten.



Met de 'positieve logica' aansluiting zal de pull down weerstand die naast de schakelaar op het printplaatje staat de pin LAAG trekken als de drukknop niet ingedrukt is. Wanneer de schakelaar gesloten wordt zal die ervoor zorgen dat de pin HOOG wordt (ingedrukt = 1).

Met de 'no external pullup' aansluiting zal de pin laag getrokken worden als de drukknop wordt ingedrukt (ingedrukt = 0). Wanneer de drukknop niet wordt ingedrukt hangt deze pin nergens aan en noemen we die 'zwevend'. We kunnen dan niet weten of er 0 of 1 wordt ingelezen en deze toestand is VERBODEN.

Nu hebben moderne microcontrollers de mogelijkheid om intern een pull-up weerstand te activeren – afzonderlijk voor elke ingangspin. We activeren die met de regel: `pinMode(_pin, INPUT_PULLUP);`
Deze aansluiting heeft als groot voordeel dat we bij elke drukknop een weerstand uitsparen, maar als nadeel dat de logica inverteert.
Wat bedoelen we met logica inverteert? :

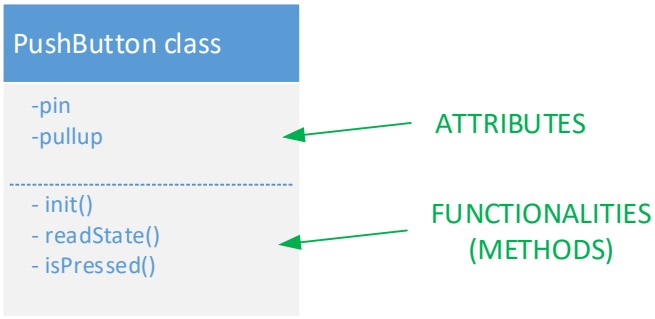
	Drukknop ingedrukt	Drukknop niet ingedrukt
POSITIEVE LOGICA	Ingelezen als 1	Ingelezen als 0
INTERNAL PULL-UP	Ingelezen als 0	Ingelezen als 1

Een ingedrukte drukknop wordt met de POSITIEVE LOGICA aansluiting ingelezen als een 1, maar indien we de drukknop aansluiten met de INTERNE PULL_UP,

Opgave: pas het programma aan zodat we van elke pin ook kunnen ingeven of er een pull-up moet ingeschakeld worden. Maak naast de “readState” nog een extra functie aan die “isPressed” noemt en die – afhankelijk van de aanwezigheid van de pull-up weerstand – effectief aangeeft of de schakelaar is ingedrukt of niet.

“readState” geeft aan of de drukknop een 1 of een 0 teruggeeft

“isPressed” geeft effectief aan of de drukknop is ingedrukt of niet



PushButton.ino

```
#include "PushButton.h"

PushButton knop(2,false);  //(pinnummer, wel of geen interne Pull-up actief)

void setup()
{
Serial.begin(9600);
knop.init();
}

void loop()
{
Serial.print(knop.readState());
Serial.print("-");
if (knop.isPressed()) Serial.println("knop is ingedrukt");
else Serial.println("knop is niet ingedrukt");
delay(100);
}
```

PushButton.h

```
// HEADER GUARDS (if Not Defined)
// GUARD zorgt ervoor dat deze code slechts 1x gedupliceerd kan worden
#ifndef PUSH_BUTTON_H
#define PUSH_BUTTON_H
```

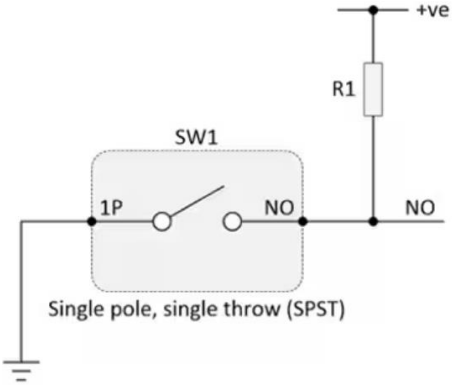
```
#include <Arduino.h>

// in*.h -> DECLARATION & INTERFACE
class PushButton
{
private:
  byte _pin;
  bool _pullUp;

public:
  // CONSTRUCTORS
  PushButton() {}
  PushButton(byte pin);
  PushButton(byte pin, bool pullUp);    //OVERLOAD CONSTRUCTOR

  // METHODS
  void init();
  bool readState();
  bool isPressed();  // bool - niet void - want isPressed geeft TRUE of FALSE
terug
};

#endif
```

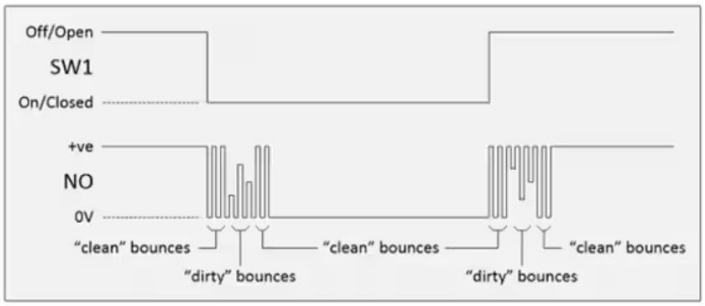
PushButton.cpp

```
#include "PushButton.h"

// in*.cpp -> IMPLEMENTATION OR LOGIC of the CLASS

// CONSTRUCTOR
PushButton::PushButton(byte pin)
{
     _pin = pin;
}

PushButton::PushButton(byte pin, bool pullUp)
{
     _pin = pin;
     _pullUp = pullUp;
}

// METHODS
```

```
void PushButton::init()
{
  if (_pullUp)pinMode(_pin, INPUT_PULLUP);
  else pinMode(_pin, INPUT);
}

bool PushButton::readState()
{
  if (digitalRead(_pin)) return true;  // return = geef terug
  else return false;
}

bool PushButton::isPressed()
{
  if(_pullUp)
  {
   if (digitalRead(_pin)) return false;
    else return true;
  }
  else {
   if (digitalRead(_pin)) return true;
    else return false;
  }

}
```

Pushbutton versie 3 met Debounce





Bounce of "Dender" bij een schakelaar treedt op wanneer de schakelaar fysiek wordt ingedrukt of losgelaten, wat kan leiden tot meerdere snelle aan/uit-overgangen. Dit kan onbedoelde signalen veroorzaken. Om dit te voorkomen, wordt debouncing toegepast, waarbij elektronica of code wordt gebruikt om deze snelle veranderingen te stabiliseren tot één enkele, betrouwbare signaalovergang. Onderstaande aanvulling op onze reeds geschreven code zorgt voor een sofwarematige ontdendering of deBouncing.

Opgave: bekijk de werking van onderstaand programma aandachtig

PushButton.ino

```
#include "PushButton.h"

PushButton knop(2,false,true);  //(pin(byte), PullUp(bool), deBounce(bool))

void setup()
{
Serial.begin(9600);
knop.init();
}

void loop()
{
Serial.print(knop.readState());
Serial.print("-");
if (knop.isPressed()) Serial.println("knop is ingedrukt");
else Serial.println("knop is niet ingedrukt");
delay(100);
}
```

PushButton.h

```
// HEADER GUARDS (if Not Defined)
// GUARD zorgt ervoor dat deze code slechts 1x gedupliceerd kan worden
#ifndef PUSH_BUTTON_H
```

```
#define PUSH_BUTTON_H

#include <Arduino.h>

// in*.h -> DECLARATION & INTERFACE
class PushButton
{
private:
  byte _pin;
  bool _pullUp;
  bool _deBounce;
  bool state = 0;
  unsigned long lastTimeStateChanged;
  unsigned long debounceDelay;

public:
  // CONSTRUCTORS
  PushButton() {}
  PushButton(byte pin);
  PushButton(byte pin, bool pullUp);    //OVERLOAD CONSTRUCTOR
  PushButton(byte pin, bool pullUp, bool deBounce);    //OVERLOAD CONSTRUCTOR

  // METHODS
  void init();
  bool readState();
  bool isPressed();  // bool - niet void - want isPressed geeft TRUE of FALSE
terug
};

#endif
```

PushButton.cpp

```
#include "PushButton.h"

// in*.cpp -> IMPLEMENTATION OR LOGIC of the CLASS

// CONSTRUCTOR
PushButton::PushButton(byte pin) {
  _pin = pin;
}

PushButton::PushButton(byte pin, bool pullUp) {
  _pin = pin;
  _pullUp = pullUp;
```

```
}

PushButton::PushButton(byte pin, bool pullUp, bool deBounce) {
  _pin = pin;
  _pullUp = pullUp;
  _deBounce = deBounce;
  lastTimeStateChanged = millis();
  debounceDelay = 50;
}

// METHODS
void PushButton::init() {
  if (_pullUp) pinMode(_pin, INPUT_PULLUP);
  else pinMode(_pin, INPUT);
  state = digitalRead(_pin);
}

bool PushButton::readState() {
  if (!(_deBounce))
  {
    state = digitalRead(_pin);
  }

  else
  {
    unsigned long timeNow = millis();
    if (timeNow - lastTimeStateChanged > debounceDelay) {
      bool newState = digitalRead(_pin);
      if (newState != state) {
        state = newState;
        lastTimeStateChanged = timeNow;
      }
    }
  }
  return state;
}

bool PushButton::isPressed() {
  readState();
  if (_pullUp) return !state;
  else return state;
}
```

Uitdaging:
Beschouw deze regel in het hoofdprogramma:

```
PushButton knop(2,false,true);  //(pin(byte), PullUp(bool), deBounce(bool))
```

Pas het programma aan zodat je met deze regel de debouncetijd van 50msec kan meegeven – als deze op 0 wordt gezet mag er geen debounce worden toegepast.

```
PushButton knop(2,false,50);  //(pin(byte), PullUp(bool), deBounce(int))
```

Pushbutton en leds samen in 1 programma

OOP_les3.ino

```
#include "PushButton.h"
#include "Led.h"

#define RedLedPin 9
#define YellowLedPin 10
#define GreenLedPin 11
#define ButtonPin 2

//3 objecten van de Led class en 1 object van de PushButton class
Led redLed(RedLedPin);
Led yellowLed(YellowLedPin);
Led greenLed(GreenLedPin);
PushButton knop(ButtonPin,false,true); //(pin(byte), PullUp(bool),
deBounce(bool))
```

```
void setup() {
  redLed.init();
  yellowLed.init();
  greenLed.init();
  knop.init();

}

void loop() {
  if (knop.isPressed())
  {
    redLed.on();
    yellowLed.off();
    greenLed.on();
  }
  else
  {
    redLed.off();
    yellowLed.on();
    greenLed.off();
  }
```

Led Blinker met overerving

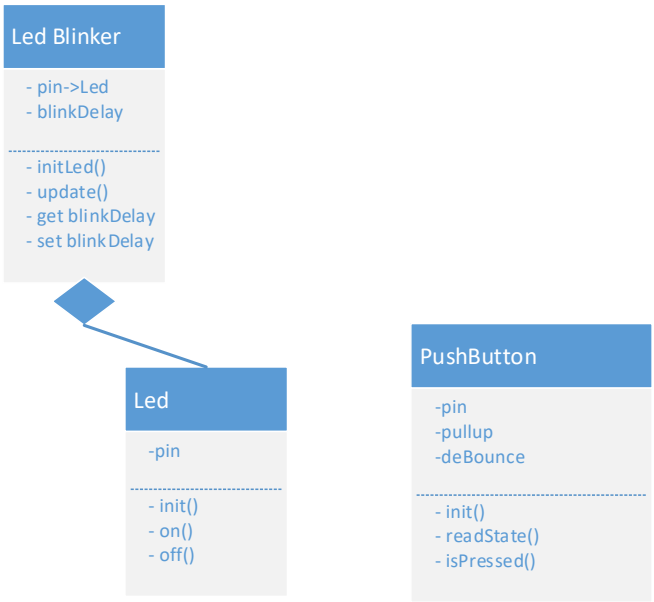
In deze les kom **OVERERVING** of 'Enheritance' voor de eerste keer aan bod. OVERERVING is een belangrijk principe van OOP.
Stel je voor dat je een 'Auto' klasse hebt. Een auto heeft eigenschappen zoals kleur, merk, model, en functies zoals starten en stoppen. Dan wil je misschien ook specifieke typen auto's hebben, zoals 'SUV', 'Sedan' en 'Hatchback'. Deze specifieke typen auto's delen veel eigenschappen en functies met de algemene 'Auto' klasse, maar hebben ook hun eigen unieke kenmerken.
Hier komt overerving om de hoek kijken. In plaats van elke keer opnieuw alle eigenschappen en functies voor elk specifiek autotype te schrijven, kun je de 'Auto' klasse als de basis nemen en deze uitbreiden voor elk specifiek type auto.

In deze les gebruiken we ook voor de eerste keer de **Getter en Setter** methode.
Bij **ENCAPSULATIE** is het belangrijk om zoveel mogelijk eigenschappen private te maken om de klasse te beschermen tegen onbedoelde wijzigingen vanuit externe code.
Door de Getter/Setter methode te gebruiken kunnen we toch op een ietwat veiligere methode toegang krijgen tot deze private variabelen van een klasse.

In deze les maken we een nieuwe klasse 'LedBlinker' die we de eigenschappen laten 'OVER-ERVEN' van de klasse 'Led'

In een UML schema ziet dat er als volgt uit: (het overervingsteken is de ruit)
Vanuit de ledBlinker klasse zal:

- De ledpin gekozen kunnen worden
- De blinkdelay ingesteld kunnen worden
- De led geïnitialiseerd kunnen worden
- Via een update 'gekeken' worden of de toestand van de led moet wijzigen
- De blinkDelay(private) toch tijdens de uitvoer van het programma uitgelezen en gewijzigd kunnen worden.

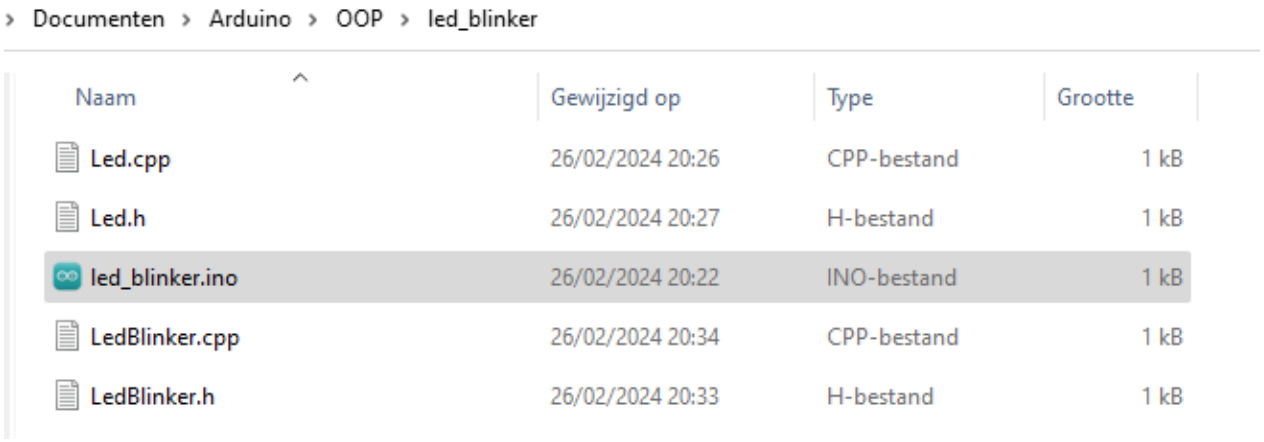


We pakken dit aan in 4 stappen:

LedBlinker STAP 1 Overerving

Doel STAP 1:

- Klasse LedBlinker aanmaken die alle eigenschappen overerft van klasse Led
- In Klasse LedBlinker een initialisatie method aanmaken



We maken 2 extra files aan: LedBlinker.cpp en LedBlinker.h

STEP 1 STEP 2 STEP 3 STEP 4

Led_blinker.ino

```
#include "Led.h"
#include "LedBlinker.h"

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9

Led redLed(RED_LED_PIN);  // Maak een object “redLed” van de klasse Led en
attribute RED_LED_PIN
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);

LedBlinker redLedBlinker(redLed); // Maak een object “redLedBlinker” van de
klasse LedBlinker en redLed (die een attribute is van de klasse Led)
LedBlinker yellowLedBlinker(yellowLed);
LedBlinker greenLedBlinker(greenLed);



void setup() {
```

```
  redLedBlinker.initLed();  // init via de LedBlinker Klasse
  yellowLedBlinker.initLed();
  greenLedBlinker.initLed();
}

void loop() {
redLed.on();     // hier gebruiken we de Led klasse
delay(500);
redLed.off();
delay(500);
}
```

Led.h

```
#ifndef LED_H
#define LED_H
#include <Arduino.h>

class Led
{
private:
  byte _pin;

public:
  Led() {} // Default constructor- specifiek nodig bij Overerving
  Led(byte pin);

  // init the pin for the LED
  // call this in setup()
  void init();
  void init(bool defaultState);

  // power on the LED
  void on();
  // power off the LED
  void off();
};

#endif
```

Led.cpp

```
#include "Led.h"
```

```
Led::Led(byte pin)
{
  _pin = pin;
}

void Led::init()
{
  pinMode(_pin, OUTPUT);
}

void Led::init(bool defaultState)
{
  init();
  if (defaultState) {
    on();
  }
  else {
    off();
  }
}

void Led::on()
{
  digitalWrite(_pin, HIGH);
}

void Led::off()
{
  digitalWrite(_pin, LOW);
}
```

LedBlinker.h

```
#ifndef LED_BLINKER_H
#define LED_BLINKER_H

#include <Arduino.h>
#include "Led.h"

class LedBlinker
{
private:
```

```
  Led _led;         // 'variabele _led van het type 'class Led'
                    // overerving van alle eigenschappen van class Led
                    // byte _led zouden we vroeger gedaan hebben
                    // hier roepen we de default constructor Led() {} aan

public:
  LedBlinker() {} // do not use
  LedBlinker(Led &led); //&led = by reference
                    //led mag ook, maar maakt copy en kost extra geheugen


  void initLed();
};

#endif
```

LedBlinker.cpp

```
#include "LedBlinker.h"

LedBlinker::LedBlinker(Led &led) // constructor
{
  _led = led;
}

void LedBlinker::initLed()
{
  _led.init();   // Functie die in classe Led zit

}
```

LedBlinker STAP 2 Toggle

Doel stap 2:

- In klasse Led: isPoweredOn() method toevoegen
- In klasse Led: toggle() method toevoegen
- In klasse LedBlinker toggleLed() method toevoegen die toggle() uit Led klasse gebruikt

STEP 1 STEP 2 STEP 3 STEP 4

Led_blinker.ino

```
#include "Led.h"
#include "LedBlinker.h"

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9

Led redLed(RED_LED_PIN);  // Maak een object “redLed” van de klasse Led en attribute RED_LED_PIN
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);

LedBlinker redLedBlinker(redLed); // Maak een object “redLedBlinker” van de klasse LedBlinker en redLed (die een attribute is van de klasse Led)
LedBlinker yellowLedBlinker(yellowLed);
LedBlinker greenLedBlinker(greenLed);

void setup() {
  redLedBlinker.initLed();  // init via de LedBlinker Klasse
  yellowLedBlinker.initLed();
  greenLedBlinker.initLed();
}

void loop() {
  redLedBlinker.toggleLed();
 // if (redLed.isPoweredOn()) werkt niet - klasse moet via overerving aangeroepen worden
  if (redLedBlinker.isPoweredOnLed()) yellowLed.off();
  else yellowLed.on();
  delay(500);
  redLedBlinker.toggleLed();
  if (redLedBlinker.isPoweredOnLed()) yellowLed.off();
```

```
  else yellowLed.on();
  delay(500);
}
```

Led.h

```
#ifndef LED_H
#define LED_H
#include <Arduino.h>

class Led
{
private:
  byte _pin;
  bool state;

public:
  Led() {} // Default constructor- specifiek nodig bij Overerving
  Led(byte pin);


  // init the pin for the LED
  // call this in setup()
  void init();
  void init(bool defaultState);
  // power on the LED
  void on();
  // power off the LED
  void off();
  // power on/off the LED depending on previous state
  void toggle();
  // check if led is on
  bool isPoweredOn();

};

#endif
```

Led.cpp

```
#include "Led.h"

Led::Led(byte pin)
{
```

```
  _pin = pin;
  state = LOW;
}

void Led::init()
{
  pinMode(_pin, OUTPUT);
}

void Led::init(bool defaultState)
{
  init();
  if (defaultState) {
    on();
  }
  else {
    off();
  }
}

void Led::on()
{
  state = HIGH;
  digitalWrite(_pin, state);
}

void Led::off()
{
  state = LOW;
  digitalWrite(_pin, state);
}

bool Led::isPoweredOn()
{
  return state;
}

void Led::toggle()
{
  if (isPoweredOn()) {
    off();
  }
  else {
    on();
  }
```

```
}
```

LedBlinker.h

```
#ifndef LED_BLINKER_H
#define LED_BLINKER_H

#include <Arduino.h>
#include "Led.h"

class LedBlinker
{
private:
  Led _led;     // ‘variabele _led van het type ‘class Led’
      // overerving van alle eigenschappen van class Led
      // hier roepen we de default constructor Led() {} aan


public:
  LedBlinker() {} // do not use
  LedBlinker(Led &led); //&led = by reference
//led mag ook, maar maakt copy en kost extra geheugen
  void initLed();
  void toggleLed();
  bool isPoweredOnLed();

};

#endif
```

LedBlinker.cpp

```
#include "LedBlinker.h"

LedBlinker::LedBlinker(Led &led) // constructor //&led = by reference
{
  _led = led;
}

void LedBlinker::initLed()
{
  _led.init();   // Functie die in classe Led zit-> let op de underscore voor
_led

}
```

```
void LedBlinker::toggleLed()
{
  _led.toggle(); // Functie die in classe Led zit-> let op de underscore voor
_led
}

bool LedBlinker::isPoweredOnLed()
{
  _led.isPoweredOn(); // Functie die in classe Led zit-> let op de underscore
voor _led
}
```

LedBlinker STAP 3 Blinkerfrequentie

Doel stap 3

- Via een 'update' method een blinkerfrequentie invoegen in de LedBlinker klasse. Gebruik hiervoor de millis() functie in Arduino.

STEP 1 STEP 2 STEP 3 STEP 4

Led_blinker.ino

```
#include "Led.h"
#include "LedBlinker.h"

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9

Led redLed(RED_LED_PIN);
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);

LedBlinker redLedBlinker(redLed, 200);

LedBlinker yellowLedBlinker(yellowLed, 450);
LedBlinker greenLedBlinker(greenLed, 899);

void setup() {
  redLedBlinker.initLed();
  yellowLedBlinker.initLed();
  greenLedBlinker.initLed();
}

void loop() {
  redLedBlinker.update();
  yellowLedBlinker.update();
  greenLedBlinker.update();
}
```

Led.h

```
#ifndef LED_H
#define LED_H
#include <Arduino.h>
```

```
class Led
{
private:
  byte _pin;
  bool state;

public:
  Led() {} // Default constructor- specifiek nodig bij Overerving
  Led(byte pin);


  // init the pin for the LED
  // call this in setup()
  void init();
  void init(bool defaultState);
  // power on the LED
  void on();
  // power off the LED
  void off();
  // power on/off the LED depending on previous state
  void toggle();
  // check if led is on
  bool isPoweredOn();

};

#endif
```

Led.cpp

```
#include "Led.h"

Led::Led(byte pin)
{
  _pin = pin;
  state = LOW;
}

void Led::init()
{
  pinMode(_pin, OUTPUT);
}
```

```
void Led::init(bool defaultState)
{
  init();
  if (defaultState) {
    on();
  }
  else {
    off();
  }
}

void Led::on()
{
  state = HIGH;
  digitalWrite(_pin, HIGH);
}

void Led::off()
{
  state = LOW;
  digitalWrite(_pin, LOW);
}

bool Led::isPoweredOn()
{
  return state;
}

void Led::toggle()
{
  if (isPoweredOn()) {
    off();
  }
  else {
    on();
  }
}
```

LedBlinker.h

```
#ifndef LED_BLINKER_H
#define LED_BLINKER_H

#include <Arduino.h>
```

```
#include "Led.h"

class LedBlinker
{
private:
  Led _led;     // ‘variabele _led van het type ‘class Led’
      // overerving van alle eigenschappen van class Led
      // hier roepen we de default constructor Led() {} aan
  unsigned long lastTimeBlinked;
  unsigned long _blinkDelay;


public:
  LedBlinker() {} // do not use
  LedBlinker(Led &led); //&led = by reference
  LedBlinker(Led &led, unsigned long blinkDelay);
  void initLed();
  void toggleLed();
  bool isPoweredOnLed();
  void update();
};

#endif
```

LedBlinker.cpp

```
#include "LedBlinker.h"

LedBlinker::LedBlinker(Led &led) // constructor //&led = by reference
{
  _led = led;
  lastTimeBlinked = millis();
  _blinkDelay = 500;


}

LedBlinker::LedBlinker(Led &led, unsigned long blinkDelay) // Overload
constructor
{
  _led = led;
  lastTimeBlinked = millis();
  _blinkDelay = blinkDelay;
}
```

```
void LedBlinker::initLed()
{
  _led.init();
}

void LedBlinker::toggleLed()
{
  _led.toggle();
}

bool LedBlinker::isPoweredOnLed()
{
  _led.isPoweredOn();
}

void LedBlinker::update()
{
  unsigned long timeNow = millis();
  if (timeNow - lastTimeBlinked > _blinkDelay) {
    lastTimeBlinked = timeNow;
    toggleLed();
  }
}
```

LedBlinker STAP 4 Getter en Setter

Doel stap 4

- We willen op elk moment in het programma kunnen opvragen wat de ingestelde delay is voor een bepaalde led. (vermits `_blinkDelay` private is kan dit niet zo maar)
- We willen op elk moment in het programma de delay van elke led kunnen aanpassen – dus niet enkel meer bij het aanroepen van de constructor. (vermits `_blinkDelay` private is kan dit niet zo maar)
- Gebruik hiervoor de getter en setter methode

STEP 1 STEP 2 STEP 3 STEP 4

Led_blinker.ino

```
#include "Led.h"
#include "LedBlinker.h"
unsigned long lastTimeHere = 0;

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9

Led redLed(RED_LED_PIN);
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);

LedBlinker redLedBlinker(redLed,50);
LedBlinker yellowLedBlinker(yellowLed,450);
LedBlinker greenLedBlinker(greenLed,899);

void setup() {
  redLedBlinker.initLed();  // init via de LedBlinker Klasse
  yellowLedBlinker.initLed();
  greenLedBlinker.initLed();
}

void loop() {
  redLedBlinker.update();

  unsigned long time = millis();  // elke seconde frequentie aanpassen
  if (time - lastTimeHere > 1000) {
    lastTimeHere = time;
    if (redLedBlinker.getBlinkDelay() < 500)
```

```
      {
        redLedBlinker.setBlinkDelay(redLedBlinker.getBlinkDelay()+50);
      }
    else redLedBlinker.setBlinkDelay(50);

  }
}
```

Led.h

```
#ifndef LED_H
#define LED_H
#include <Arduino.h>

class Led
{
private:
  byte _pin;
  bool state;

public:
  Led() {} // Default constructor- specifiek nodig bij Overerving
  Led(byte pin);
  void init();
  void init(bool defaultState);
  void on();
  void off();
  void toggle();
  bool isPoweredOn();

};

#endif
```

Led.cpp

```
#include "Led.h"

Led::Led(byte pin)
{
  _pin = pin;
  state = LOW;
}
```

```
void Led::init()
{
  pinMode(_pin, OUTPUT);
}

void Led::init(bool defaultState)
{
  init();
  if (defaultState) {
    on();
  }
  else {
    off();
  }
}

void Led::on()
{
  state = HIGH;
  digitalWrite(_pin, HIGH);
}

void Led::off()
{
  state = LOW;
  digitalWrite(_pin, LOW);
}

bool Led::isPoweredOn()
{
  return state;
}

void Led::toggle()
{
  if (isPoweredOn()) {
    off();
  }
  else {
    on();
  }
}
```

LedBlinker.h

```
#ifndef LED_BLINKER_H
#define LED_BLINKER_H

#include <Arduino.h>
#include "Led.h"

class LedBlinker
{
private:
  Led _led;     // ‘variabele _led van het type ‘class Led’
      // overerving van alle eigenschappen van class Led
      // hier roepen we de default constructor Led() {} aan
  unsigned long lastTimeBlinked;
  unsigned long _blinkDelay;


public:
  LedBlinker() {} // do not use
  LedBlinker(Led &led); //&led = by reference
  LedBlinker(Led &led, unsigned long blinkDelay);
  void initLed();
  void toggleLed();
  bool isPoweredOnLed();
  void update();
  unsigned long getBlinkDelay();
  void setBlinkDelay(unsigned long blinkDelay);

};

#endif
```

LedBlinker.cpp

```
#include "LedBlinker.h"

LedBlinker::LedBlinker(Led &led) // constructor //&led = by reference
{
  _led = led;
  lastTimeBlinked = millis();
  _blinkDelay = 500;


}
```

```
LedBlinker::LedBlinker(Led &led, unsigned long blinkDelay) // Overload
constructor
{
  _led = led;
  lastTimeBlinked = millis();
  _blinkDelay = blinkDelay;
}


void LedBlinker::initLed()
{
  _led.init();   // Functie die in classe Led zit-> let op de underscore voor
_led

}

void LedBlinker::toggleLed()
{
  _led.toggle(); // Functie die in classe Led zit-> let op de underscore voor
_led
}

bool LedBlinker::isPoweredOnLed()
{
  _led.isPoweredOn(); // Functie die in classe Led zit-> let op de underscore
voor _led
}

void LedBlinker::update()
{
  unsigned long timeNow = millis();
  if (timeNow - lastTimeBlinked > _blinkDelay) {
    lastTimeBlinked = timeNow;
    toggleLed();
  }
}

// GETTER METHODE
unsigned long LedBlinker::getBlinkDelay()
{
  return _blinkDelay;
}

// SETTER METHODE
void LedBlinker::setBlinkDelay(unsigned long blinkDelay)
```

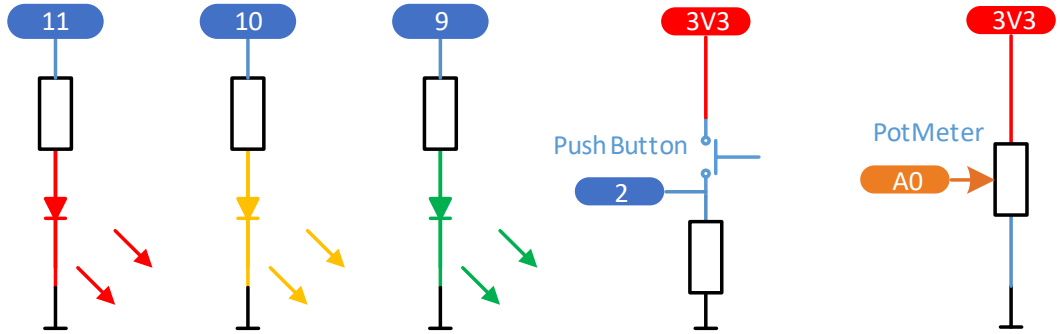
```
{
  _blinkDelay = blinkDelay;
}
```

Extra Uitdagingen LedBlinker:

1. met drukknop klasse frequentie beïnvloeden
2. hoog en laagtijd variëren met 2 meegegeven attributes
3. PWM signalen genereren met attributes frequentie en DutyCycle

Traffic Light

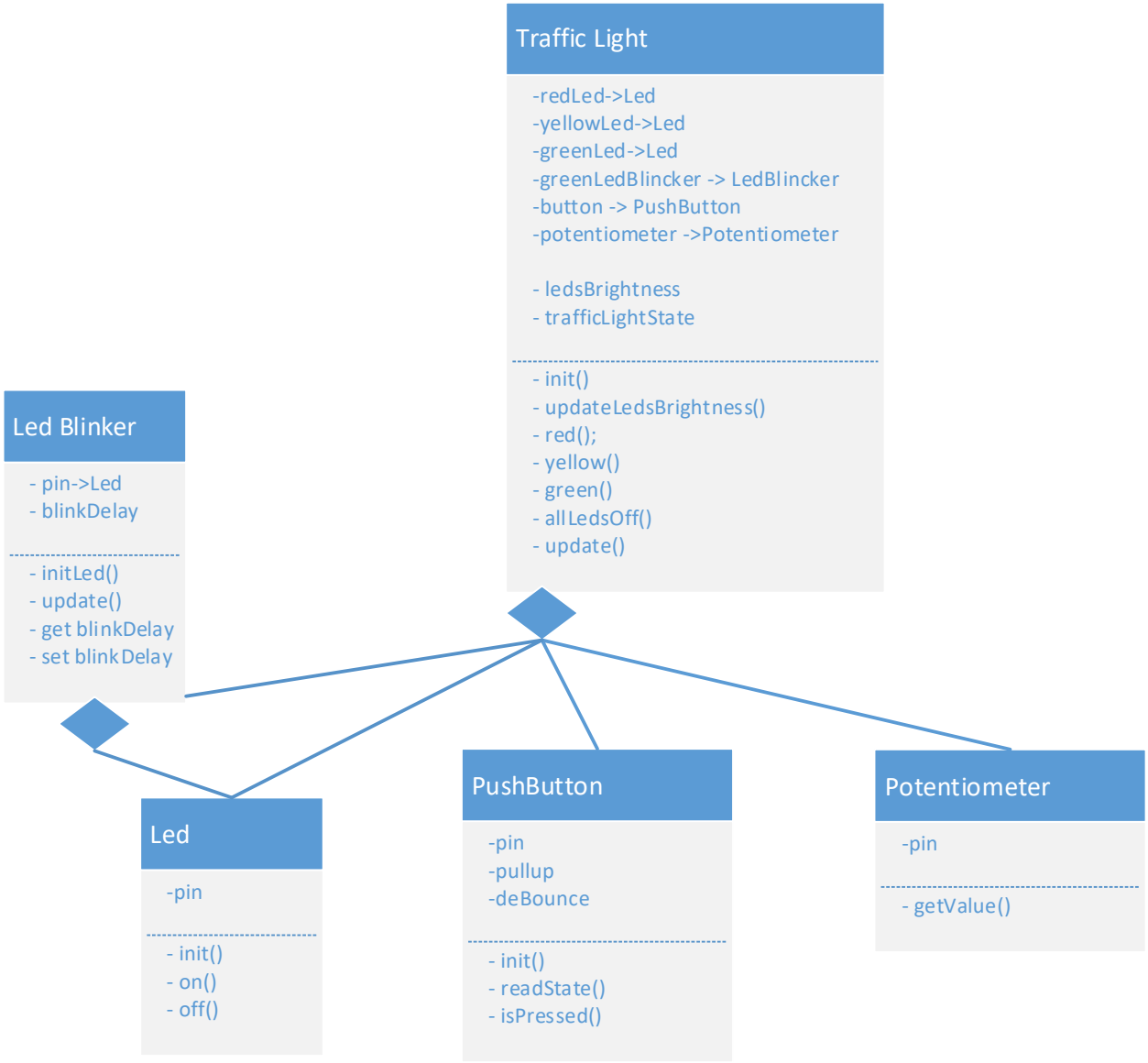
Schema



Het uiteindelijk doel van het traffic light system is om het volgende te doen

- De rode led brandt in rust – dit is de toestand waarin het systeem altijd terugkomt
- Van het moment op de drukknop gedrukt wordt
    - Gaat de gele led aan gedurende 2sec
    - Daarna gaat de gele led uit
    - En gaat de groene led stabiel aan gedurende 2sec
    - Daarna gaat de groene led knipperen gedurende 3sec
    - Dan gaat de groene led terug uit
    - EN gaat de rode led terug aan
- Met de potentiometer kunnen we instellen hoe hard de 3 leds gedimd zijn

UML voorstelling van het afgewerkte project



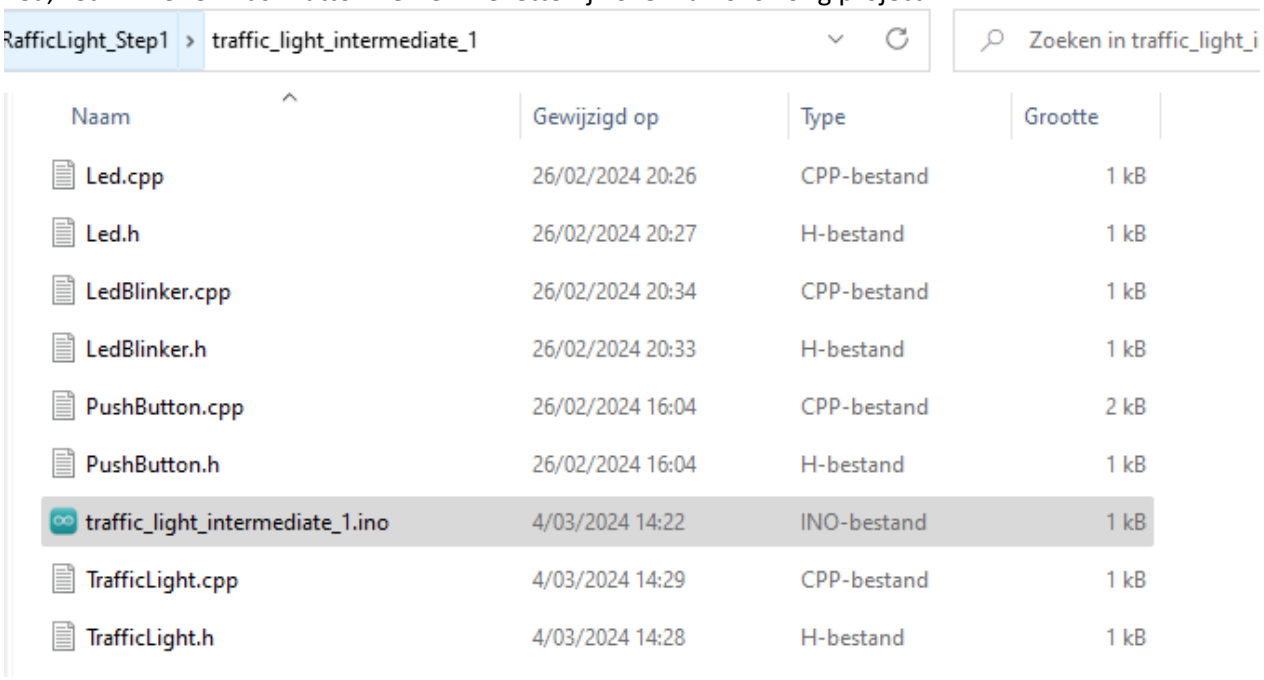
We pakken dit stap voor stap aan:

Traffic Light step 1 Overerving

Opgave

- Maak een TrafficLight klasse aan die OVERERFT van Led, LedBlinker en PushButton.
- Met TrafficLight.init moeten alle leds en de drukknop geïnitialiseerd worden
- Met TrafficLight.red mag enkel de rode led branden
- Met TrafficLight.yellow mag enkel de gele led branden
- Met TrafficLight.green mag enkel de groen led branden
- Met TrafficLight. allLedsOff mag geen enkele led branden

Led, LedBlinker en PushButton nemen we letterlijk over van ons vorig project



We maken een nieuwe .h en .cpp TrafficLight –file bij.

TrafficLight.h

```
#ifndef TRAFFIC_LIGHT_H
#define TRAFFIC_LIGHT_H

#include "Led.h"
#include "LedBlinker.h"
#include "PushButton.h"

class TrafficLight
{
private:
  Led _redLed;     // we roepen 3 objecten aan van de class 'Led' -> OVERERVING
  Led _yellowLed;
```

```
  Led _greenLed;
  LedBlinker _greenLedBlinker; // OVERERVING van class ‘LedBlinker’
  PushButton _button;          // OVERERVING van class ‘PushButton’

public:
  TrafficLight() {} // do not use
  TrafficLight(
    Led &redLed, Led &yellowLed, Led &greenLed,
    LedBlinker &greenLedBlinker, PushButton &button); // Constructor van
samengestelde OVERERVING classes

  void init();

  void red();
  void yellow();
  void green();
  void allLedsOff();
};

#endif
```

TrafficLight.cpp

```
#include "TrafficLight.h"

TrafficLight::TrafficLight(
                Led &redLed, Led &yellowLed, Led &greenLed,
                LedBlinker &greenLedBlinker, PushButton &button)
{
  _redLed = redLed;
  _yellowLed = yellowLed;
  _greenLed = greenLed;
  _greenLedBlinker = greenLedBlinker;
  _button = button;
}

void TrafficLight::init()
{
  _redLed.init();  // aanroepen init functie in de verschillende Child classen
  _yellowLed.init();
  _greenLed.init();
  _button.init();
  delay(100);
```

```
  red();            //startsituatie – rode led aan.
}

void TrafficLight::red()
{
  _redLed.on();
  _yellowLed.off();
  _greenLed.off();
}

void TrafficLight::yellow()
{
  _redLed.off();
  _yellowLed.on();
  _greenLed.off();
}

void TrafficLight::green()
{
  _redLed.off();
  _yellowLed.off();
  _greenLed.on();
}

void TrafficLight::allLedsOff()
{
  _redLed.off();
  _yellowLed.off();
  _greenLed.off();
}
```

Traffic_Light STEP1.ino

We initiëren alle klassen en runnen een eenvoudig testprogramma

```
#include "Led.h"
#include "LedBlinker.h"
#include "PushButton.h"
#include "TrafficLight.h"

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9
#define BUTTON_PIN 2
```

```
Led redLed(RED_LED_PIN); // functie uit klasse Led
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);
LedBlinker greenLedBlinker(greenLed, 100); // object uit child klasse Led
PushButton button(BUTTON_PIN, true, true);  // functie uit klasse Pushbutton
TrafficLight trafficLight(
      redLed, yellowLed, greenLed,
      greenLedBlinker, button);

void setup() {
  trafficLight.init(); // slechts 1 init om alles te initialiseren
}

void loop() {
  trafficLight.red();
  delay(500);
  trafficLight.yellow();
  delay(500);
  trafficLight.green();
  delay(500);
  trafficLight.allLedsOff();
  delay(500);
}
```

Traffic Light step 2 – StateMachine

Opgave:

- Implementeer de 4 toestanden of states (rode led aan, gele led aan, groene led aan, groene led knipperen) in de 'TrafficLight' klasse
- Gebruik hiervoor best een **state machine**
- Gebruik hiervoor best een **switch case** programmeerstructuur
- Om de leesbaarheid te bevorderen gebruik je best een **enum** declaratie om logische namen te gebruiken ipv state nummers.

Enum (Enumeratie):

Een enum is een manier om symbolische namen te definiëren voor een reeks gehele getallen. In dit geval definiëren we LEDkleur als een enum met de opties Rood, Groen en Blauw. Elk van deze opties staat eigenlijk voor een getal (0 voor Rood, 1 voor Groen, en 2 voor Blauw), maar we gebruiken de namen om het leesbaarder te maken.

Voorbeeldprogramma enum

```
enum LEDkleur {
  Rood,
  Groen,
  Blauw
};

void setup() {
  Serial.begin(9600);

  LEDkleur mijnLED = Groen;

  if (mijnLED == Rood) {
    Serial.println("De LED is rood!");
  } else if (mijnLED == Groen) {
    Serial.println("De LED is groen!");
  } else {
    Serial.println("De LED is blauw!");
  }
}
void loop() {  // Hier gebeurt niets in dit voorbeeld
}
```

Switch case programmeerstructuur

De switch-case instructie is een programmeerstructuur zoals de if else, while en for structuren. De switch case wordt gebruikt om een blok code uit te voeren op basis van de exacte waarde van een variabele. Het is een handige manier om verschillende acties uit te voeren, afhankelijk van de waarde van een enkele variabele, en het helpt om de code overzichtelijk te houden.

! Merk wel op dat dit mogelijk is maar dit **niet**

```
case 2:
case >2:
```

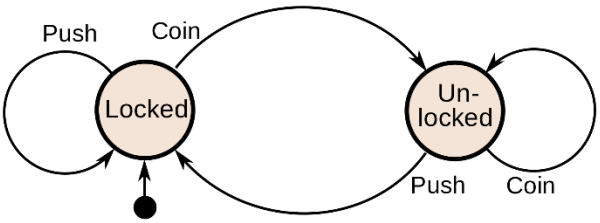
Voorbeeldprogramma switch case

```
int optie = 2;

void setup() {
  Serial.begin(9600);
}

void loop() {
  switch(optie) {
    case 1:
      Serial.println("Optie 1 geselecteerd.");
      break;
    case 2:
      Serial.println("Optie 2 geselecteerd.");
      break;
    case 3:
      Serial.println("Optie 3 geselecteerd.");
      break;
    default:
      Serial.println("Ongeldige optie!");
      break;
  }

  delay(1000);
}
```

State Machine

Een state machine (toestandsmachine) is een model voor het beschrijven van het gedrag van een systeem door zijn verschillende toestanden en overgangen tussen die toestanden te definiëren. Het wordt vaak gebruikt in software-ontwikkeling om de logica van een programma te structureren en complexe systemen gemakkelijker te begrijpen en te implementeren.

In essentie beschrijft een state machine een systeem dat zich in verschillende toestanden kan bevinden, waarbij elke toestand bepaalt hoe het systeem reageert op invoer (input) en welke acties het uitvoert. Wanneer een bepaalde gebeurtenis plaatsvindt, kan het systeem van de ene toestand naar de andere overgaan volgens vooraf gedefinieerde overgangsregels.



Een voorbeeld van een eenvoudig mechanisme dat gemodelleerd kan worden door een toestandsmachine is een draaipoortje.

Beschouwd als een state-machine heeft een draaipoortje twee mogelijke toestanden: Locked en Unlocked. Er zijn twee mogelijke invoeren die van invloed zijn op de toestand: het plaatsen van een munt in de gleuf (coin) en het duwen van de arm (push). In de vergrendelde toestand heeft duwen op de arm geen effect; ongeacht hoe vaak de invoer push wordt gegeven, blijft deze in de vergrendelde toestand. Het plaatsen van een munt - dat wil zeggen, het geven van de machine een muntinvoer - verschuift de toestand van Locked naar Unlocked. In de ontgrendelde toestand heeft het plaatsen van extra munten geen effect; dat wil zeggen, het geven van extra muntinvoeren verandert de toestand niet. Een klant die door de armen duwt, geeft een push-invoer en reset de toestand naar Locked.

Draaipoort programma sequentieel:

```
#define COIN_PIN 2   // Pin voor de muntinvoer
#define PUSH_PIN 3   // Pin voor de armduw-invoer
#define ARM_PIN 4    // Pin voor de armuitvoer

bool locked = true;  // Begint vergrendeld

void setup() {
  pinMode(COIN_PIN, INPUT_PULLUP);  // Muntinvoerpin als invoer met pull-up
weerstand
  pinMode(PUSH_PIN, INPUT_PULLUP);  // Armduwinvoerpin als invoer met pull-up
weerstand
  pinMode(ARM_PIN, OUTPUT);          // Armpin als uitvoer
}

void loop() {
  if (locked) {
    if (digitalRead(COIN_PIN) == LOW) {
      unlockTurnstile();
    }
  } else {
    if (digitalRead(PUSH_PIN) == LOW) {
      lockTurnstile();
    }
  }
}

void unlockTurnstile() {
  digitalWrite(ARM_PIN, HIGH);  // Draai de arm open
  delay(1000);  // Wacht enkele seconden voor het doorlaten van de klant
  digitalWrite(ARM_PIN, LOW);   // Draai de arm terug
  locked = false;  // Schakel de vergrendelde status uit
}

void lockTurnstile() {
  digitalWrite(ARM_PIN, LOW);  // Draai de arm terug
  locked = true;  // Schakel de vergrendelde status in
}
```

Draaipoort programma STATEMACHINE:

```
#define COIN_PIN 2   // Pin voor de muntinvoer
#define PUSH_PIN 3   // Pin voor de armduw-invoer
#define ARM_PIN 4    // Pin voor de armuitvoer

enum State { LOCKED, UNLOCKED };

State currentState = LOCKED;

void setup() {
  pinMode(COIN_PIN, INPUT_PULLUP);  // Muntinvoerpin als invoer met pull-up
weerstand
  pinMode(PUSH_PIN, INPUT_PULLUP);  // Armduwinvoerpin als invoer met pull-up
weerstand
  pinMode(ARM_PIN, OUTPUT);          // Armpin als uitvoer
}

void loop() {
  switch (currentState) {
    case LOCKED:
      if (digitalRead(COIN_PIN) == LOW) {
        unlockTurnstile();
      }
      break;
    case UNLOCKED:
      if (digitalRead(PUSH_PIN) == LOW) {
        lockTurnstile();
      }
      break;
  }
}

void unlockTurnstile() {
  digitalWrite(ARM_PIN, HIGH);  // Draai de arm open
  delay(1000);  // Wacht enkele seconden voor het doorlaten van de klant
  digitalWrite(ARM_PIN, LOW);   // Draai de arm terug
  currentState = UNLOCKED;  // Schakel over naar ontgrendelde toestand
}

void lockTurnstile() {
  digitalWrite(ARM_PIN, LOW);  // Draai de arm terug
  currentState = LOCKED;  // Schakel over naar vergrendelde toestand
}
```

TrafficLight.h

```
#ifndef TRAFFIC_LIGHT_H
#define TRAFFIC_LIGHT_H

#include "Led.h"
#include "LedBlinker.h"
#include "PushButton.h"

class TrafficLight
{
private:
  Led _redLed;
  Led _yellowLed;
  Led _greenLed;
  LedBlinker _greenLedBlinker;
  PushButton _button;

  enum trafficLightState {
    STATE_RED,
    STATE_YELLOW,
    STATE_GREEN,
    STATE_BLINK_GREEN
  };                              //Enum instructie om logische namen te kunnen
                                  // gebruiken ipv getallen.


  int state;       // om de getallen – zie enum hierboven  – in te onthouden

  unsigned long yellowStateStartTime;
  unsigned long greenStateStartTime;
  unsigned long blinkGreenStateStartTime;

  const int yellowStateDuration = 3000;
  const int greenStateDuration = 2000;
  const int blinkGreenStateDuration = 3000;

public:
  TrafficLight() {} // do not use
  TrafficLight(
    Led &redLed, Led &yellowLed, Led &greenLed,
    LedBlinker &greenLedBlinker, PushButton &button);

  void init();
```

```
  void red();
  void yellow();
  void green();
  void allLedsOff();

  void update();
};

#endif
```

TrafficLight.cpp

```
#include "TrafficLight.h"

TrafficLight::TrafficLight(
                Led &redLed, Led &yellowLed, Led &greenLed,
                LedBlinker &greenLedBlinker, PushButton &button)
{
  _redLed = redLed;
  _yellowLed = yellowLed;
  _greenLed = greenLed;
  _greenLedBlinker = greenLedBlinker;
  _button = button;
}

void TrafficLight::init()
{
  _redLed.init();
  _yellowLed.init();
  _greenLed.init();
  _button.init();
  delay(100);

  red();
}

void TrafficLight::red()
{
  _redLed.on();
  _yellowLed.off();
  _greenLed.off();
```

```
}

void TrafficLight::yellow()
{
  _redLed.off();
  _yellowLed.on();
  _greenLed.off();
}

void TrafficLight::green()
{
  _redLed.off();
  _yellowLed.off();
  _greenLed.on();
}

void TrafficLight::allLedsOff()
{
  _redLed.off();
  _yellowLed.off();
  _greenLed.off();
}

void TrafficLight::update()     // volledig NIEUW STATE MACHINE
                                // maakt gebruik van de switch case structuur
                                // wordt elke (…usec) aangeroepen….
{
  unsigned long timeNow = millis();

  switch (state) {
    case STATE_RED: {
      red();
      if (_button.isPressed()) {
        state = STATE_YELLOW;
        yellowStateStartTime = timeNow;
      }
      break;
    }
    case STATE_YELLOW: {
      yellow();
      if (timeNow - yellowStateStartTime > yellowStateDuration) {
        state = STATE_GREEN;
        greenStateStartTime = timeNow;
      }
      break;
```

```
    }
    case STATE_GREEN: {
      green();
      if (timeNow - greenStateStartTime > greenStateDuration) {
        state = STATE_BLINK_GREEN;
        allLedsOff();
        blinkGreenStateStartTime = timeNow;
      }
      break;
    }
    case STATE_BLINK_GREEN: {
      _greenLedBlinker.update();
      if (timeNow - blinkGreenStateStartTime >
                    blinkGreenStateDuration) {
        state = STATE_RED;
      }
      break;
    }
    default: {
      state = STATE_RED;
    }
  }
}
```

TrafficLight STEP 2 STATE Machine.ino

```
#include "Led.h"
#include "LedBlinker.h"
#include "PushButton.h"
#include "TrafficLight.h"

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9
#define BUTTON_PIN 2

Led redLed(RED_LED_PIN);
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);
LedBlinker greenLedBlinker(greenLed, 100);
PushButton button(BUTTON_PIN, false, true);
TrafficLight trafficLight(
      redLed, yellowLed, greenLed,
```

```
      greenLedBlinker, button);

void setup() {
  trafficLight.init();
}

void loop() {
  trafficLight.update();
}
```

Traffic Light step 3 – Potmeter

Opgave

- Schrijf een klasse (cpp en h file) om de waarde van een potmeter op een bepaalde pin in te lezen met de instructie `getValue()`
- Pas de Led klasse (zowel cpp als h file) aan om de PWM functie op de betreffende pin te kunnen gebruiken met een waarde "brightness"

```
void Led::on(byte brightness)
{
  state = HIGH;
  analogWrite(_pin, brightness);
}
```

- Pas de trafficlight klasse aan (zowel cpp als h file) zodat de waarde van de potmeter de brightness van de 3 leds zal beïnvloeden

Potentiometer.h

```
#ifndef POTENTIOMETER_H
#define POTENTIOMETER_H

#include <Arduino.h>

class Potentiometer
{
private:
  byte _pin;

public:
  Potentiometer() {} // do not use
  Potentiometer(byte pin);

  int getValue();
};

#endif
```

Potentiometer.cpp

```
#include "Potentiometer.h"

Potentiometer::Potentiometer(byte pin)
{
```

```
  _pin = pin;
}

int Potentiometer::getValue()
{
  return analogRead(_pin);
}
```

Led.h

```
#ifndef LED_H
#define LED_H

#include <Arduino.h>

class Led
{
private:
  byte _pin;
  bool state;
public:
  Led() {} // do not use
  Led(byte pin);

  // init the pin for the LED
  // call this in setup()
  void init();
  void init(bool defaultState);

  // power on the LED
  void on();
  void on(byte brightness);
  // power off the LED
  void off();
  // power on/off the LED depending on previous state
  void toggle();
  // check if led is on
  bool isPoweredOn();
};

#endif
```

Led.cpp

```
#include "Led.h"

Led::Led(byte pin)
{
  _pin = pin;
  state = LOW;
}

void Led::init()
{
  pinMode(_pin, OUTPUT);
}

void Led::init(bool defaultState)
{
  init();
  if (defaultState) {
    on();
  }
  else {
    off();
  }
}

void Led::on()
{
  state = HIGH;
  pinMode(_pin, OUTPUT);    // om pin na PWM functie terug als IO pin te zetten
  digitalWrite(_pin, state);
}

void Led::off()
{
  state = LOW;
  pinMode(_pin, OUTPUT);      // om pin na PWM functie terug als IO pin te zetten
  digitalWrite(_pin, state);
}

void Led::on(byte brightness)
{
  state = HIGH;
  analogWrite(_pin, brightness);
}

bool Led::isPoweredOn()
```

```
{
  return state;
}

void Led::toggle()
{
  if (isPoweredOn()) {
    off();
  }
  else {
    on();
  }
}
```

TrafficLight.h

```
#ifndef TRAFFIC_LIGHT_H
#define TRAFFIC_LIGHT_H

#include <Arduino.h>

#include "Led.h"
#include "LedBlinker.h"
#include "PushButton.h"
#include "Potentiometer.h"  // include Potentiometer - OVERERVING


class TrafficLight
{
private:
  Led _redLed;
  Led _yellowLed;
  Led _greenLed;
  LedBlinker _greenLedBlinker;
  PushButton _button;
  Potentiometer _potentiometer;        // toevoegen

  byte _ledsBrightness;                // toevoegen - moet byte zijn voor
AnalogWrite

  enum trafficLightState {
    STATE_RED,
    STATE_YELLOW,
```

```
    STATE_GREEN,
    STATE_BLINK_GREEN
  };

  int state;

  unsigned long yellowStateStartTime;
  unsigned long greenStateStartTime;
  unsigned long blinkGreenStateStartTime;

  const int yellowStateDuration = 3000;
  const int greenStateDuration = 2000;
  const int blinkGreenStateDuration = 3000;

    void updateLedsBrightness();  // toegevoegd

public:
  TrafficLight() {} // do not use
  TrafficLight(
    Led &redLed, Led &yellowLed, Led &greenLed,
    LedBlinker &greenLedBlinker, PushButton &button,
    Potentiometer &potentiometer);    // toevoegen

  void init();

  void red();
  void yellow();
  void green();
  void allLedsOff();

  void update();
};

#endif
```

Trafficlight.cpp

```
#include "TrafficLight.h"

TrafficLight::TrafficLight(
                Led &redLed, Led &yellowLed, Led &greenLed,
                LedBlinker &greenLedBlinker, PushButton &button,
                Potentiometer &potentiometer)   // toevoegen
```

```
{
  _redLed = redLed;
  _yellowLed = yellowLed;
  _greenLed = greenLed;
  _greenLedBlinker = greenLedBlinker;
  _button = button;
  _potentiometer  = potentiometer;     //toevoegen
}

void TrafficLight::init()
{
  _redLed.init();
  _yellowLed.init();
  _greenLed.init();
  _button.init();
  delay(100);

  updateLedsBrightness();    //toegevoegd
  state = STATE_RED;         // toegevoegd
  red();
}

void TrafficLight::red()
{
  _redLed.on(_ledsBrightness);   //
  _yellowLed.off();
  _greenLed.off();
}

void TrafficLight::yellow()
{
  _redLed.off();
  _yellowLed.on(_ledsBrightness);    //
  _greenLed.off();
}

void TrafficLight::green()
{
  _redLed.off();
  _yellowLed.off();
  _greenLed.on(_ledsBrightness);     //
}

void TrafficLight::allLedsOff()
{
```

```
  _redLed.off();
  _yellowLed.off();
  _greenLed.off();
}

void TrafficLight::updateLedsBrightness()     // Toegevoegd
{
  // 0..1023 -> 0..255
  _ledsBrightness = _potentiometer.getValue() / 4;
}

void TrafficLight::update()
{
  updateLedsBrightness();          // aangepast
  unsigned long timeNow = millis();

  switch (state) {
    case STATE_RED: {
      red();
      if (_button.isPressed()) {
        state = STATE_YELLOW;
        yellowStateStartTime = timeNow;
      }
      break;
    }
    case STATE_YELLOW: {
      yellow();
      if (timeNow - yellowStateStartTime > yellowStateDuration) {
        state = STATE_GREEN;
        greenStateStartTime = timeNow;
      }
      break;
    }
    case STATE_GREEN: {
      green();
      if (timeNow - greenStateStartTime > greenStateDuration) {
        state = STATE_BLINK_GREEN;
        allLedsOff();
        blinkGreenStateStartTime = timeNow;
      }
      break;
    }
    case STATE_BLINK_GREEN: {
      _greenLedBlinker.update();
      if (timeNow - blinkGreenStateStartTime >
```

```
                    blinkGreenStateDuration) {
        state = STATE_RED;
      }
      break;
    }
    default: {
      state = STATE_RED;
    }
  }
}
```

Final program.ino

```
#include "Led.h"
#include "LedBlinker.h"
#include "PushButton.h"
#include "TrafficLight.h"
#include "Potentiometer.h"    //Aangepast

#define RED_LED_PIN 11
#define YELLOW_LED_PIN 10
#define GREEN_LED_PIN 9
#define BUTTON_PIN 2
#define POTENTIOMETER_PIN A0  //Aangepast

Potentiometer potentiometer(POTENTIOMETER_PIN); //Aangepast
Led redLed(RED_LED_PIN);
Led yellowLed(YELLOW_LED_PIN);
Led greenLed(GREEN_LED_PIN);
LedBlinker greenLedBlinker(greenLed, 100);
PushButton button(BUTTON_PIN, false, false);
TrafficLight trafficLight(
      redLed, yellowLed, greenLed,
      greenLedBlinker, button, potentiometer); // aangepast

void setup() {
  trafficLight.init();
}

void loop() {
  trafficLight.update();
}
```