Embedded Systems C++

Datum und Uhrzeitzähler mit Objektorientierung auf dem STM32F401

Projektarbeit

des Studienganges Elektrotechnik – Infotronik

an der Dualen Hochschule Baden-Württemberg Mosbach

von

Niklas Linsler, Ben Meyer, Lorena Volk, Tom Wagenpfahl

16.06.2023

Matrikelnummer, Kurs Niklas Linsler: 3674806, MOS-TET20B

Ben Meyer: 8073820, MOS-TET20B

Lorena Volk: 3479898, MOS-TET20B

Tom Wagenpfahl: 5240692, MOS TET20B

Gutachter\*in der Dualen Hochschule Dipl.-Ing.(DH) Claudia Heß

Erklärung

Wir versichern hiermit, dass wir unsere Projektarbeit mit dem Thema: „Embedded Systems C++ Datum und Uhrzeitzähler mit Objektorientierung auf dem STM32F401“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Mosbach, 16.06.2023

Ort, Datum Unterschrift Niklas Linsler

Mosbach, 16.06.2023

Ort, Datum Unterschrift Ben Meyer

Mosbach, 16.06.2023

Ort, Datum Unterschrift Lorena Volk

Mosbach, 16.06.2023

Ort, Datum Unterschrift Tom Wagenpfahl

Inhaltsverzeichnis

[Erklärung 2](#_Toc136774094)

[Inhaltsverzeichnis 3](#_Toc136774095)

[1. Vorwort 4](#_Toc136774096)

[2. Klassendiagramm 5](#_Toc136774097)

[3. Sequenzdiagramm 6](#_Toc136774098)

[4. Klassenbeschreibung 7](#_Toc136774099)

[4.1 HMI 7](#_Toc136774100)

[4.2 ConsoleHMI 9](#_Toc136774101)

[4.3 HardwareHMI 11](#_Toc136774102)

[4.4 DateTime 16](#_Toc136774103)

[4.5 Counter 20](#_Toc136774104)

[4.6 ModuloCounter 21](#_Toc136774105)

[4.7 OffsetCounter 23](#_Toc136774106)

[5. Schluss 25](#_Toc136774107)

# Vorwort

# Ein Bild, das Text, Diagramm, parallel, Plan enthält. Automatisch generierte BeschreibungKlassendiagramm

# Sequenzdiagramm

Ein Bild, das Text, Reihe, Screenshot, Diagramm enthält.

Automatisch generierte Beschreibung

# Klassenbeschreibung

## HMI

Ein Bild, das Text, Screenshot, Schrift, Symbol enthält.

Automatisch generierte Beschreibung

HMI (Human Machine Interface) ist ein Interface. Das soll die Eingaben des Benutzers mit dem Programm verbinden. Es wird als abstrakter Typ jedes HMIs verwendet. Die Funktionen werden hier als Grundlage für die Klassen „ConsoleHMI“ und „HardwareHMI“ definiert. In diesen Klassen sind sie benutzt auch Implementiert. Das Interface selbst hat keine Implementierung und besteht nur aus virtuellen Methoden. Daher besteht hier nur eine Header- und keine CPP-Datei. Außerdem ist das HMI als Singleton ausgeführt. Es kann also nur eine Instanz der Klasse vorhanden seine.

### Public Operations:

virtual bool HMI::checkRun()

Gibt den Wert True zurück, wenn der Zähler in diesem Schritt um eine Sekunde inkrementiert werden soll.

virtual bool HMI::checkShow()

Prüft, ob die Zeit angezeigt werden soll, gibt True zurück, wenn dies der Fall ist.

static HMI & HMI::getInstance()

Holt eine Singleton-Instanz des HMI‘s. Ist noch keine Instanz angelegt, wird diese neu angelegt.

Rückgabewert:

Referenz auf das HMI (HMI&)

virtual void HMI::showDate(int day,int month, int year )

Zeigt das aktuelle Datum an.

Attribute:

day: Tag des Monats als Ganzzahl

month: Monat des Jahres als Ganzzahl

year: Das aktuelle Jahr als Ganzzahl

virtual void HMI::showDateTime(int day, int month, int year, int hour, int minute, int second)

Zeigt das aktuelle Datum und die aktuelle Zeit an.

Attribute:

day: Tag des Monats als Ganzzahl

month: Monat des Jahres als Ganzzahl

year: Das aktuelle Jahr als Ganzzahl

hour: Die aktuelle Stunde als Ganzzahl

minute: Die aktuelle Minute als Ganzzahl

second: Die aktuelle Sekunde als Ganzzahl

virtual void HMI::showTime(int hour, int minute, int second)

Zeigt die aktuelle Zeit an.

Attribute:

hour: Die aktuelle Stunde als Ganzzahl

minute: Die aktuelle Minute als Ganzzahl

second: Die aktuelle Sekunde als Ganzzahl

## ConsoleHMI

Ein Bild, das Text, Symbol, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Die Klasse ConsoleHMI dient als HMI (Human Machine Interface) zwischen dem Menschen und der Eingabekonsole. Sie erbt von der Klasse „HMI“ und implementiert die Funktionen dieser Klasse. Sie ist auch als Singleton-Pattern implementiert, da nur eine Instanz der Eingabekonsole vorhanden sein soll.

### Private Attributes:

int32\_t runNumber

Gibt an, wie oft noch hochgezählt werden soll. Das ist wichtig, damit mehr als eine Sekunde als Zählung nach jeder Eingabe durch den Nutzer möglich werden.

### Private Operations:

ConsoleHMI::ConsoleHMI()

Konstruiert ein Eingebekonsolen-HMI als ein Singleton-Objekt, da nur eine Eingebekonsole vorhanden sein soll.

ConsoleHMI::ConsoleHMI(const ConsoleHMI&) = delete

Löscht den Kopier-Konstruktor, damit kein zweites Objekt des ConsoleHMI erzeugt werden kann. Gibt damit außerdem Speicherplatz frei.

void ConsoleHMI::operator=(const ConsoleHMI &) = delete

Löscht den Zuweisungsoperator, damit kein weiteres Objekt des ConsoleHMI erzeugt werden kann. Gibt damit außerdem Speicherplatz frei.

### Public Operations:

virtual bool HMI::checkRun ()

Sollte in einer Schleife aufgerufen werden und gibt False zurück, wenn runNumber auf „0“ steht. Gibt True zurück, wenn runNumber noch größer als 0 ist.

Wurde noch kein Wert für runNumber eingegeben, oder der vorherige Wert für runNumber ist auf „0“, wird nach einer Eingabe des Wertes für runNumber durch den Benutzer gefragt. Nach Eingeben des Wertes wird von diesem noch Eins abgezogen und der Rückgabewert auf True gestellt. Ist der Eingegebene Wert nicht größer als Null, wird False zurückgegeben.

Gibt es bereits einen Wert für runNumber, wird dieser um Eins verringert und True zurückgegeben.

bool ConsoleHMI::checkShow()

Sollte in einer Schleife aufgerufen werden. Ist das Ende einer Inkrementierungsserie erreicht, dann wird True zurückgegeben. Damit kann dann das aktuelle Datum ausgegeben werden. So wird möglich, dass auf der Konsole nur das Datum nach einem Durchlauf angezeigt wird.

static ConsoleHMI & ConsoleHMI::getInstance()

Holt die Singleton-Instanz des ConsoleHMI´s.

Rückgabewert:

Referenz auf das ConsoleHMI

void ConsoleHMI::showDate(int day,int month, int year)

Gibt das Datum im Format [DD.MM.YYYY] aus.

Attribute:

day: Tag des Monats als Ganzzahl

month: Monat des Jahres als Ganzzahl

year: Das aktuelle Jahr als Ganzzahl

void ConsoleHMI::showDateTime(int day, int month, int year, int hour, int minute, int second)

Gibt das Datum und die Uhrzeit im Format [DD.MM.YYYY–HH:MM:SS] aus.

Attribute:

day: Tag des Monats als Ganzzahl

month: Monat des Jahres als Ganzzahl

year: Das aktuelle Jahr als Ganzzahl

hour: Die aktuelle Stunde als Ganzzahl

minute: Die aktuelle Minute als Ganzzahl

second: Die aktuelle Sekunde als Ganzzahl

void ConsoleHMI::showTime (int hour, int minute, int second)

Gibt die Zeit im Format [HH:MM:SS] aus.

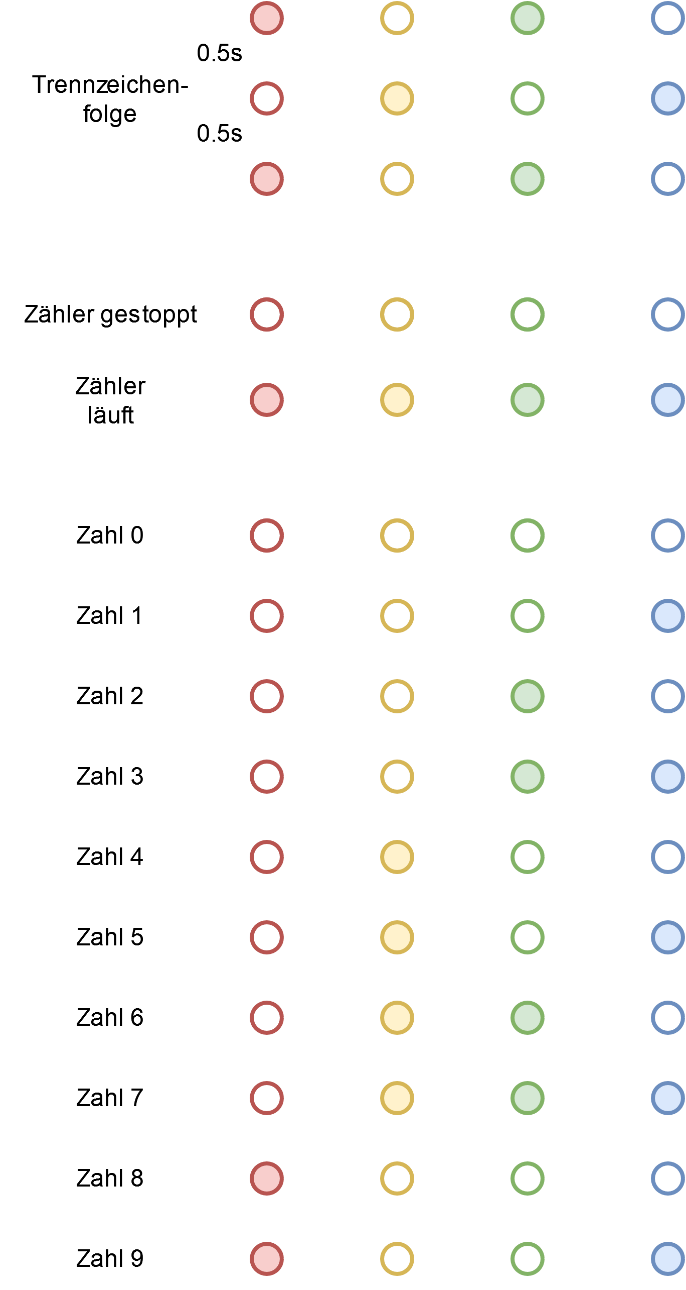
Attribute:

hour: Die aktuelle Stunde als Ganzzahl

minute: Die aktuelle Minute als Ganzzahl

second: Die aktuelle Sekunde als Ganzzahl

## HardwareHMI

Ein Bild, das Text, Symbol, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Die Klasse HardwareHMI bildet das HMI (Human Machine Interface) zwischen dem Benutzer und der verwendeten Hardware. Sie erbt von der Klasse HMI und ist ebenfalls ein Singleton-Pattern, da das Programm nur auf einer Hardware läuft.

Ein Bild, das Screenshot, Farbigkeit, Licht enthält.

Automatisch generierte BeschreibungDie Darstellung der LED Bedeutungen in der   
HardwareHMI-Klasse ist auf der vorherigen Seite und links dargestellt.

### Private Attributes:

StmGpio gpioA

Speichert das GPIO-Objekt von Port A

StmGpio gpioB

Speichert das GPIO-Objekt von Port A

bool isRunning

Boolean, welcher Speichert, ob das Programm läuft oder gestoppt ist. Standardbelegung ist False, also dass das Programm nicht läuft.

static const int LED\_1\_A

Speichert den Pin von Port A, an welchem LED 1 ist.

Der Standard-Pin ist Pin 8.

static const int LED\_2\_A

Speichert den Pin von Port A, an welchem LED 2 ist.

Der Standard-Pin ist Pin 9.

static const int LED\_3\_A

Speichert den Pin von Port A, an welchem LED 3 ist.

Der Standard-Pin ist Pin 10.

static const int LED\_4\_B

Speichert den Pin von Port B, an welchem LED 4 ist.

Der Standard-Pin ist Pin 10.

static const int SW\_1\_B

Speichert den Pin von Port B, an welchem der Button 1 ist.

Der Standard-Pin ist Pin 5.

static const int SW\_2\_B

Speichert den Pin von Port B, an welchem der Button 2 ist.

Der Standard-Pin ist Pin 3.

Static const uint32\_t GPIOA\_ADDRESS

Speichert die Adresse von Port A auf dem STM32F401.

Die Standard-Adresse ist 0x40020000.

static const uint32\_t GPIOB\_ADDRESS

Speichert die Adresse von Port B auf dem STM32F401.

Die Standard-Adresse ist 0x40020400.

### Private Operations:

HardwareHMI ()

Konstruktor des HardwareHMIs. Das HardwareHMI ist die Ausgabe auf den STM32 Ports und ist als Singleton umgesetzt.

Im Konstruktor werden die Pins über die Funktion setupHardware() initialisiert.

HardwareHMI (const HardwareHMI &) = delete

Löscht den Kopier-Konstruktor, damit kein zweites Objekt des ConsoleHMI erzeugt werden kann. Gibt damit außerdem Speicherplatz frei.

void operator = (const HardwareHMI &) = delete

Löscht den Zuweisungsoperator, damit kein zweites Objekt des ConsoleHMI erzeugt werden kann. Gibt damit außerdem Speicherplatz frei.

void setupHardware ()

In setupHardware werden die verwendeten Ports der Hardware initialisiert. Es wird auf die Objekte gpioA und gpioB der gegebenen Klasse stm\_gpio zugegriffen, um die Hardware nutzen zu können.

LED 1 (rot) an D7 / PA8

LED 2 (gelb) an D8 / PA9

LED 3 (grün) an D2 / PA10

LED 4 (blau) an D3 / PB3

SWITCH 1 an D4 / PB5

SWITCH 2 an D6 / PB6

void setLEDs (bool led\_1, bool led\_2, bool led\_3, bool led\_4)

Die 4 LEDs werden mit Bool-Werten gesetzt.

Attribute:

led\_1: Status der LED 1. True: an; False: aus

led\_2: Status der LED 2. True: an; False: aus

led\_3: Status der LED 3. True: an; False: aus

led\_4: Status der LED 4. True: an; False: aus

void sleep ()

Das System pausiert (schläft) für eine kurze Zeit. Diese Zeit wird durch einen hochzählenden Counter gemessen.

void sleepLong ()

Das System pausiert (schläft) für eine lange Zeit (Faktor 10 des kurzen Schlafs). Diese Zeit wird durch einen hochzählenden Counter gemessen.

void showNumber (int number)

Die mitgegebene Ziffer wird auf den LEDs angezeigt. Hierfür wird die Ziffer binär dargestellt und an „SetLEDs“ übergeben.

Attribute:

number: Die anzuzeigende Ziffer als Ganzzahl.

### Public Operations:

static HardwareHMI & HardwareHMI::getInstance()

Holt die Singleton-Instanz des HardwareHMI´s

Rückgabewert:

Referenz auf das HardwareHMI

void HardwareHMI::showDate(int day, int month, int year)

Zeigt das Datum auf der Hardware wie folgt an: (Codiert nach der Abbildung die am Beginn dieser Klasse gezeigt ist.

Trennung

Tage Zehner Stelle

Tage Einer Stelle

Trennung

Monate Zehner Stelle

Monate Einer Stelle

Trennung

Jahre Tausender Stelle

Jahre Hunderter Stelle

Jahre Zehner Stelle

Jahre Einer Stelle

Trennung

Attribute:

day: Tag des Monats als Ganzzahl

month: Monat des Jahres als Ganzzahl

year: Das aktuelle Jahr als Ganzzahl

void HardwareHMI::showTime(int hour, int minute, int second)

Zeigt die Zeit auf der Hardware wie folgt an: (Codiert nach der Abbildung die am Beginn dieser Klasse gezeigt ist.

Trennung

Stunde Zehner Stelle

Stunde Einer Stelle

Trennung

Minute Zehner Stelle

Minute Einer Stelle

Trennung

Sekunde Zehner Stelle

Sekunde Einer Stelle

Trennung

Attribute:

hour: Die aktuelle Stunde als Ganzzahl

minute: Die aktuelle Minute als Ganzzahl

second: Die aktuelle Sekunde als Ganzzahl

void HardwareHMI::showDateTime(int day, int month, int year, int hour, int minute, int second)

Ruft die Operationen showDate() und showTime() auf um das Datum und die Uhrzeit auszugeben.

bool HardwareHMI::checkRun()

Gibt True zurück, wenn der Run-Button (SW\_1\_B) gedrückt wurde. Gibt False zurück, wenn er nicht gedrückt wurde. Dies lässt sich durch erneutes Drücken des Tasters rückgängig machen. Die Taste ist entprellt und muss deshalb ca. 0,5 Sekunden gedrückt werden.

bool HardwareHMI::checkShow()

Gibt True zurück, wenn der Show-Button (SW\_2\_B) gedrückt wurde. Gibt False zurück, wenn er nicht gedrückt wurde. Die Taste ist entprellt und muss deshalb ca. 0,5 Sekunden gedrückt werden.

## DateTime

Die Klasse „DateTime“ ist der Mittelpunkt der Anwendung. Alle weiteren Klassen sind über Komposition oder Komposition und Vererbung von dieser Klasse ausgehend. In der „main“ wird die Funktion „run“ des Objekts „dateTime“ der Klasse „DateTime“ in einer Endlosschleife aufgerufen.

### Private Attributes:

Counter year

Der Counter, der die Jahre zählt.

OffsetCounter month

Der Counter, der die Monate zählt.

Als „OffsetCounter“ ist es möglich, den aktuellen Monat immer wieder bei 1 (also Januar) statt 0 starten zu lassen.

OffsetCounter day

Der Counter, der die Tage zählt.

Als „OffsetCounter“ ist es möglich, den aktuellen Tag immer wieder bei 1 statt bei 0 starten zu lassen.

ModuloCounter hour

Der Counter, der die Stunden zählt.

Der „ModuloCounter“ lässt dabei die Stundenanzahl immer wieder bei 0 starten.

ModuloCounter minute

Der Counter, der die Minuten zählt.

Der „ModuloCounter“ lässt dabei die Minutenanzahl immer wieder bei 0 starten.

ModuloCounter second

Der Counter, der die Sekunden zählt.

Der „ModuloCounter“ lässt dabei die Sekundenanzahl immer wieder bei 0 starten.

uint32\_t updateDaysCounter

Mit dem 32 Bit langen unsigned Int werden Sekunden gezählt.

Es dient zum Vergleich mit dem Minimalwert „SECONDS\_FOR\_SMALLEST\_MONTH“ beim Updaten der Tage im Monat.

### Static Private Attributes:

static HMI & hmi = ConsoleHMI::getInstance  
oder ￼static HMI & hmi = HardwareHMI::getInstance()

Das HMI kann entweder durch die Konsole oder die Hardware eingebunden sein.

static const uint32\_t SECONDS\_FOR\_SMALLEST\_MONTH = 2419200

Die minimale Anzahl an Sekunden in einem Monat. Damit wird die Anzahl der Tage in diesem Monat nur 1 Mal pro Monat erneuert und nicht jede Sekunde.

Der Vergleichswert für „updateDaysCounter“.

static const int LEAP\_YEAR\_ARRAY [12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

In diesem Array sind alle Tage pro Monat für ein Schaltjahr hinterlegt.

static const int NON\_LEAP\_YEAR\_ARRAY [12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

In diesem Array sind alle Tage pro Monat für ein Nicht-Schaltjahr hinterlegt.

### Private Operations:

void countSecond()

Inkrementiert die Sekunden um 1.

Prüft außerdem, ob bereits die minimale Anzahl an Sekunden für einen Monat abgelaufen sind und aktualisiert gegebenenfalls den Stoppwert für das Day-Objekt.

bool istLeapYear(int year=2000)

Berechnet, ob das aktuelle Jahr ein Schaltjahr ist.

Attribute: Das aktuelle Jahr.

Rückgabewert: True, wenn ein Schaltjahr ist.

int getMonthDays(int thisMonth)

Ermittelt die Anzahl der Tage des aktuellen Monats und gibt diese zurück. Dafür wird auf „LEAP\_YEAR\_ARRAY“ und „NON\_LEAP\_YEAR\_ARRAY“ zugegriffen.

Rückgabewert: Anzahl der Tage dieses Monats

### Public Operations:

DateTime (int startSecond=0, int startMinute=0, int startHour=0, int startDay=1, int startMonth=1, int startYear=2000)

Der Konstruktor für ein neues DateTime-Objekt.

Attribute:

startSecond: Der Startwert für die Sekunden. Wird kein Startwert mitgegeben, wird der Wert für die aktuelle Sekunde auf 0 gesetzt.

startMinute: Der Startwert für die Minuten. Wird kein Startwert mitgegeben, wird der Wert für die aktuelle Minute auf 0 gesetzt.

startHour: Der Startwert für die Stunden. Wird kein Startwert mitgegeben, wird der Wert für die aktuelle Stunde auf 0 gesetzt.

startDay: Der Startwert für den Tag. Wird kein Startwert mitgegeben, wird der Wert für den aktuellen Tag auf 1 gesetzt.

startMonth: Der Startwert für den Monat. Wird kein Startwert mitgegeben, wird der Wert für den aktuellen Monat auf 1 gesetzt.

startYear: Der Startwert für das Jahr. Wird kein Startwert mitgegeben, wird der Wert für das aktuelle Jahr auf 2000 gesetzt.

void run()

Durch den Aufruf der „run“-Funktion wird der Counter laufen gelassen. Jeder Aufruf entspricht einem Inkrement der Sekunde.

Das Inkrement wird durch einen Aufruf der „countSecond()“ Funktion umgesetzt.

## Counter

Ein Bild, das Text, Schrift, Screenshot, Symbol enthält.

Automatisch generierte Beschreibung

Klasse mit einem einfachen Zähler der geschrieben, gelesen und um eins erhöht werden kann

### Private Attributes:

int count:

Zähler als Ganzzahl

### Public Operations:

explicit Counter(int startCount = 0) : count(startCount)

Konstruktor erstellt eine Instanz von Counter mit einem Startwert auf den der Zähler (count) gesetzt wird. Der Konstruktor wird nicht kopiert. Konstruktor mit Initialisierungsliste.

Attribute:

startCount: Wert auf den der Zähler gesetzt wird als Ganzzahl

virtual int Counter::getCount()

Gibt den aktuellen Zählwert als Ganzzahl zurück

void Counter::setCount(int newCount)

Setzt den Zähler (count) auf die angegebene Ganzzahl (newCount).

* + - * Attribute:
        + int newCount: Der neue Zählwert für den Zähler (Count) als Ganzzahl

virtual void Counter::increment()

* + - Zählt den Wert des Zählers (count) um einen Wert hoch.

## ModuloCounter

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

* Modulo Counter erbt von der Klasse Counter. Sie dient zum Bestimmen wann der nächste Zähler hochgezählt werden soll und der Zähler wieder zurückgesetzt werden soll.

### Private Attributes:

Counter\* nextCounter:

Zeiger auf den nächsten Zähler (Counter), nach Erreichen des maximalen Wertes eines Zählers wird der Zähler am Zeiger um eins erhöht.

int moduloValue:

Wert durch den geteilt wird als Ganzzahl mit Standardwert 100.

### Public Operations:

ModuloCounter(Counter \*nextModuloCounter = nullptr, int startValue = 0, int moduloValue = 100) : Counter(startValue), nextCounter(nextModuloCounter), moduloValue(moduloValue){}

* + - Konstruktor mit Festlegung für den Zeiger auf den nächsten Zähler, für den Startwert und den Modulowert. Konstruktor mit Initialisierungsliste.
      * Attribute:
        + Counter \*nextModuloCounter = nullptr: Zeiger auf den nächsten Counter nach Erreichen des maximalen Zählwertes
        + int startValue = 0: Startwert für den Zähler
        + int moduloValue: Wert mit dem der Modulo gebildet wird

virtual void increment()

* + - Zähler um einen Wert vergrößern

void setModuloValue(int newValue)

* + - setzt den Modulowert auf einen anderen Wert
      * Attribute:
        + int newValue: neuer Wert für den Modulo

int getModuloValue()

* + - gibt den aktuellen Modulowert zurück

## OffsetCounter

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

* Klasse zum Erstellen eines Offsets für die Monate und Tage im Jahr. Erbt von der Klasse ModuloCounter und somit auch von der Klasse Counter

### Private Attributes:

int startCountingAt:

Offsetwert für die Offsetcounter

int stopCountingAt:

Endwert für den Modulozähler

### Public Operations:

OffsetCounter(Counter \*nextCounter = nullptr, int startValue = 1, int startCountingAt = 1, int stopCountingAt = 30) : ModuloCounter( nextCounter, startValue - startCountingAt, stopCountingAt - startCountingAt + 1), startCountingAt(startCountingAt), stopCountingAt(stopCountingAt){}

* + - Konstruktor zum Anlegen eines OffsetCounter. Konstruktor mit Initialisierungsliste.
      * Attribute:
        + Counter \*nextCounter = nullptr: Zeiger auf den nächsthöheren Zähler
        + int startValue = 1: Setzt den Startwert für den Zähler fest
        + int startCountingAt = 1: Offsetwert für die Zähler
        + int stopCountingAt = 30: Modulowert für den Zähler

void increment()

* + - Zähler um einen Wert erhöhen

int getCount()

* + - gibt den aktuellen Zählwert zurück

void setStopCountingAt(int newStopCountingAt)

* + - setzt den Modulowert auf den Wert
      * Attribut:
        + newStopCountingAt: neuer Wert für den Modulozähler

int getStopCountingAt()

* + - gibt den Wert für den letzten Zählerstand wieder, bevor wieder zurückgesetzt wird

# Schluss