Stellen Sie bitte sicher, dass alles wie vorhergesehen läuft, bevor Sie dieses Übungsblatt abgeben. **Starten Sie den Kernel neu** (in der Menüleiste die Option Kernel→Restart auswählen) und **validieren** Sie anschließend das Übungsblatt (in der Menüleiste auf Validate klicken) um Rückmeldung zu eventuellen fehlenden oder fehlerhaften Eingaben zu erhalten.

Füllen Sie alle Stellen im Übungsblatt aus, welche entweder `DEIN CODE HIER` oder "DEINE ANTWORT HIER" enthalten. Geben Sie unterhalb Ihren vollständigen Namen an.

Wenn Sie Code-Bestandteile aus anderen Quellen (wie z.B. Stackoverflow) kopieren, dann machen sie den kopierten Code in ihrer Quellcodedatei kenntlich und fügen eine Referenz auf die Quelle als Kommentar hinzu.

Wenn Sie die Aufgaben in einer Gruppe erledigen, dann fügen Sie die Namen aller Gruppenmitglieder in der nachfolgende Zelle zu `Name` und zusätzlich als Kommentar am Anfang Ihrer Quellcodedatei hinzu.

In [ ]:

```
NAME = [redacted]
```

DIE UNTERNEHMERISCHE HOCHSCHULE

Methoden der Softwareentwicklung II

SS 2021

Praktische Klausur

Freitag, 09. April 2021

Roboter

Es soll die Steuersoftware eines Roboters programmiert werden. Dieser Roboter besitzt einen Motor, der für die Fortbewegung sorgt, und mehrere verschiedene Sensoren, die auf Hindernisse und Gefahren in der Umgebung achten. Wenn ein Hindernis oder eine Gefahr erkannt wurde, dann soll der Motor je nach Schwere der Gefahr entweder langsamer laufen oder sofort gestoppt werden.

Das Herzstück der Steuerungssoftware ist die sog. Event-Loop (https://de.wikipedia.org/wiki/Ereignisschleife). Diese ist eine Endlosschleife, die ständig die Sensoren abfrägt (d.h. die `checkSensor()` -Funktion der Sensoren aufruft) und je nach Rückgabewert der Sensoren die Geschwindigkeit der Motoren steuert.

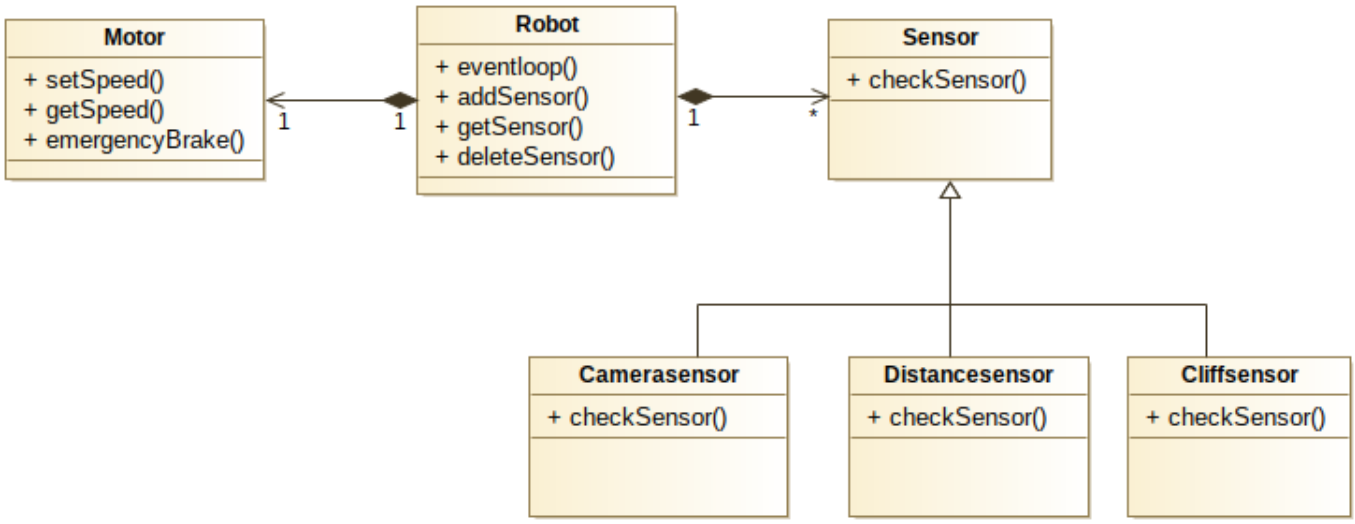
Ablauf Event-Loop

Die Event-Loop fragt ständig in einer Endlosschleife den Status aller Sensoren ab. Die Rückgabewerte der Sensoren ist eine Zahl zwischen 0 und 9, je nach Schwere der entdeckten Gefahr (0 bedeutet keine Gefahr und 9 ist die höchste Gefahrenstufe). Nachdem alle Sensoren abgefragt wurden, wird die höchste gemeldete Gefahrenstufe verwendet um die Geschwindigkeit des Motors zu setzen. Zum Beispiel, bei keiner gemeldeten Gefahr kann der Roboter sich mit voller Geschwindigkeit bewegen und beim höchsten Gefahrenleven nur mit niedrigster Geschwindkeit. Wenn eine `CriticalDangerException` geworfen wird, soll ein Notstopp der Motoren eingeleitet werden. Wenn einer der Sensoren eine `InternalErrorException` wirft, dann soll aus Sicherheitsgründen auf die niedrigste Geschwindigkeit geschalten werden.

Lassen Sie aus praktischen Gründen nach jeder Iteration die Event-Loop mindestens eine Sekunde schlafen (mit `sleep()` (https://man7.org/linux/man-pages/man3/sleep.3.html)). Wenn es zu einem Notstopp kommt oder aufgrund von internen Fehlern der Sensoren die Geschwindigkeit reduziert wurde, dann gilt dieser Zustand für 5 Iterationen bevor wieder zum Normalzustand zurückgekehrt wird. Bei Geschwindigkeitsreduktion augrund interner Sensorfehler kann es aber immer noch zur Notabschaltung aufgrund kritischer Gefahr kommen!

Grundlegende Softwarearchitektur

Die grundlegende Software-Architektur schaut folgendermaßen aus:



Public Interface

Das im folgenden gegebene öffentliche Interface soll genauso wie angegebene implementiert werden. Sie können dieses auch um weitere Funktionen erweitern, wenn Sie dies für nötig erachten.

Klasse Robot

Die Klasse `Robot` repräsentiert den Roboter selber und implementiert die zentrale Steuerung des Roboters. Sie verwaltet den Motor und die Sensoren, d.h. sie übernimmt für diese die Object Ownership. Es gibt genau einen Motor, diesen können Sie statisch zur Klasse hinzufügen (als Objektvariable). Die Sensorkonfiguration

hingegen ist ständigen Änderungen unterworfen, daher soll deren Verwaltung über einen dynamische Datencontainer erfolgen. Wählen Sie selbstständig einen passenden Datencontainer aus.

- Objektfunktionen:
  - `int addSensor(Sensor* sensor)` : Fügt einen neuen Sensor zum Roboter hinzu. Der Rückgabewert ist eine ID, die den jeweiligen Sensor **eindeutig** identifiziert.
  - `Sensor* getSensor(int id)` : Gibt den Sensor mit der angegebenen ID zurück.
  - `void deleteSensor(int id)` : Entfernt den Sensor mit der angegebenen ID und gibt alle damit verbundenen Ressourcen wieder frei.
  - `void eventLoop(int iterations)` : Diese Funktion ist das Herzstück der Steuerung, welche in einer Endlosschleife ständig den Status der Sensoren abfrägt und je nach Rückgabewert die Geschwindigkeit der Motoren einstellt. Der Eingabeparameter `iterations` gibt an, nach wievielen Iterationen die Event-Loop abgebrochen werden soll (0 bedeutet, dass die Schleife nie abgebrochen wird).

Klasse Sensor

Die Klasse `Sensor` ist die Oberklasse aller Sensoren und definiert deren öffentliches Interface.

- Objektfunktionen:
  - `int checkSensor()` : Überprüft den Status des Sensors und gibt eine positive Zahl zwischen 0 und 9 zurück, die die Einschätzung des Sensors bezüglich Gefahren in der Umgebung repräsentiert. 0 bedeutet keine Gefahr und 9 bedeutet sehr große Gefahr. Wenn eine kritische Gefahr droht (z.B. ein unmittelbarer Zusammenstoß mit einem Hindernis), dann soll eine `CriticalDangerException` geworfen werden. Wenn es zu einer Fehlfunktion des Sensors kommt, dann soll eine `InternalErrorException` geworfen werden.

Konkrete Sensoren

Es gibt drei konkrete Sensoren:

- Abstandssensor (Distancesensor)
  - Kann nur Gefahrenlevels zwischen 0 und 6 melden (wird zufällig ausgewählt)
  - Hat eine 15% Wahrscheinlichkeit eine `CriticalDangerException` zu werfen.
  - Hat eine 5% Wahrscheinlichkeit eine `InternalErrorException` zu werfen.
- Abgrundsensor (Cliffsensor)
  - Kann nur Gefahrenlevels zwischen 0 und 4 melden (wird zufällig ausgewählt)
  - Hat eine 10% Wahrscheinlichkeit eine `CriticalDangerException` zu werfen.
  - Hat eine 5% Wahrscheinlichkeit eine `InternalErrorException` zu werfen.
- Kamerasensor (Camerasensor)
  - Kann Gefahrenlevels zwischen 0 und 9 melden (wird zufällig ausgewählt)
  - Hat eine 20% Wahrscheinlichkeit eine `CriticalDangerException` zu werfen.
  - Hat eine 15% Wahrscheinlichkeit eine `InternalErrorException` zu werfen.

Klasse Motor

Die Klasse `Motor` repräsentiert den Motor. Es gibt nur einen Motortypen, daher ist hier keine Vererbungshierarchie notwendig.

- Objektfunktionen:
  - `void setSpeed(int speed)` : Setzt die Motorgeschwindigkeit. Muss eine positive Zahl zwischen 0 und 10 sein. 0 bedeutet Motorenstop und 10 bedeutet volle Geschwindigkeit.

- `int getSpeed()` : Gibt die aktuelle Geschwindigkeit zurück.
- `void emergencyBrake` : Führt einen Notstop der Motoren durch. Setzt die Geschwindigkeit auf 0.

Exceptions

Die Basisklasse aller Exceptions soll `RobotException` heißen, die wiederum von `std::exception` erbt. Überschreiben Sie die Funktion `const char* what()` der Klasse `std::exception` , sodass eine aussagekräftige Fehlermeldung zurückgegeben wird (alternativ können Sie auch von `std::runtime_error` erben und die Fehlermeldung dessen Konstruktor übergeben).

Wenn Sie es für notwendig erachten, können Sie auch weitere Exception-Klassen neben den schon oben genannten hinzufügen.

Interne Implementierung

Wie Sie die gewünschte Funktionalität intern implementieren, bleibt Ihnen überlassen. Wählen Sie passende Sichtbarkeiten und vergessen Sie nicht, wenn möglich das `const` Keyword bei Eingabeparametern und wenn passend Call-by-Reference zu verwenden.

Wählen Sie selbstständig passende Datencontainer aus. Es darf zu keinem Zeitpunkt zu einem Speicher- oder sonstigen Ressourcenleck kommen. Sorgen Sie dafür, dass, wenn die Roboter-Instanz zerstört wird, alle Ressourcen wieder sauber freigegeben werden.

Überlegen Sie sich außerdem, wo sie dynamische Bindung verwenden.

Separieren Sie bitte die `main()` Funktion in einer eigenen Datei, die Sie `main.cpp` nennen. Der Hintergrund ist, dass die automatischen Tests ihre eigene `main()` Funktion verwenden, daher muss diese einfach austauschbar sein. Der Inhalt ihrer `main()` -Funktion ist für die Prüfung nicht relevant.

Sorgen Sie für ausreichend Debug-Ausgaben bei der Programmausführung, damit man nachvollziehen kann, was der Roboter gerade tut.

Die restlichen Dateien können Sie benennen, wie Sie wollen.

Geben Sie mit der Abgabe auch ein Makefile ab, welches Ihr Programm kompiliert.

Unverbindliche Schritt-für-Schritt Anleitung

1. Beginnen Sie mit der Implementierung der Klasse `Motor` . Testen Sie diese ausführlich.
2. Beginnen Sie mit der Implementierung der Klasse `Sensor` und deren Subklassen. Gehen Sie dabei den Vererbungsbaum von oben nach unten durch. Testen Sie diese ausführlich.
3. Implementieren Sie anschließend die Klasse `Robot` und testen diese ausführlich.

In [ ]:

```
%%bash
#
# Kopieren Sie vor der Abgabe ihr Makefile und Ihre Quellcodedatein in dasselbe Verzeichnis
# in dem dieser Übungszettel zu finden ist.
#
# Der Name der ausführbaren Datei sollte 'robot' lauten.
#

make

# DEINE ANTWORT HIER
```