Bring ideas to life
VIA University College

From Design to C – Part II

ESW1

From Design to C – Part II

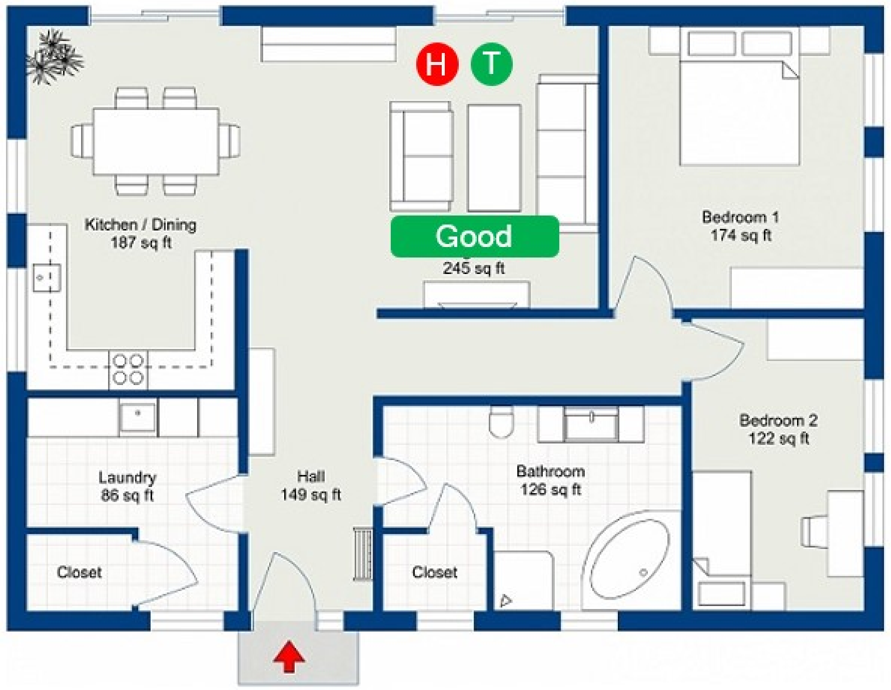
In Part I we looked at this situation:

How to come from a Design (UML) to C-code

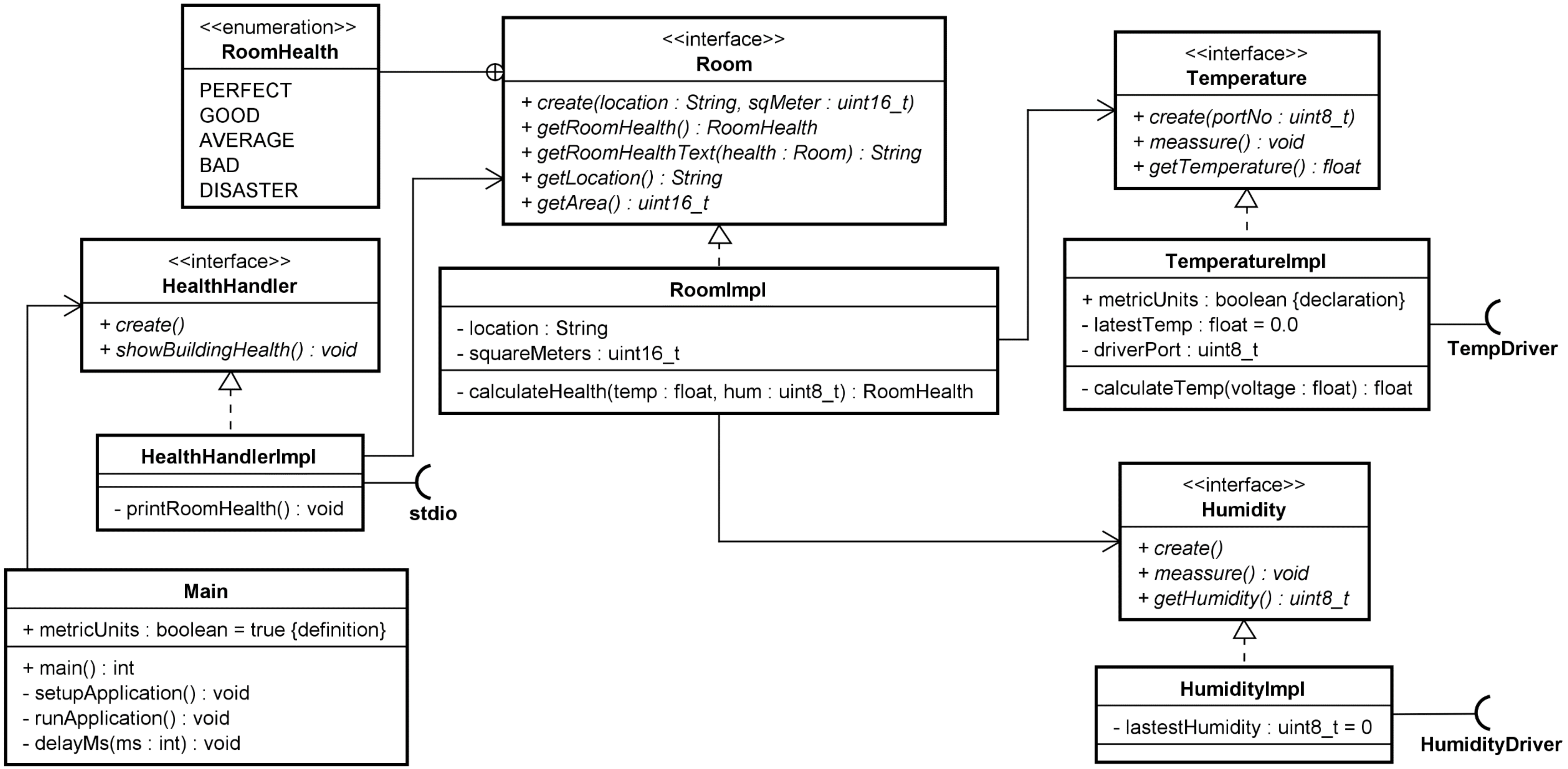
The following design is for a small application that should measure and show the environment (health) in a single room (living room) in a building

*The health is based on a **temperature-** and a **humidity-measuring** in the room. The health should be shown once per second*

The sensors are simulated in this example



From Design to C – Part I Diagram



<<enumeration>>
RoomHealth
PERFECT
GOOD
AVERAGE
BAD
DISASTER
<<interface>>
Room
+ create(location : String, sqMeter : uint16_t)
+ getRoomHealth() : RoomHealth
+ getRoomHealthText(health : Room) : String
+ getLocation() : String
+ getArea() : uint16_t
<<interface>>
Temperature
+ create(portNo : uint8_t)
+ meassure() : void
+ getTemperature() : float
<<interface>>
HealthHandler
+ create()
+ showBuildingHealth() : void
RoomImpl
- location : String
- squareMeters : uint16_t
- calculateHealth(temp : float, hum : uint8_t) : RoomHealth
TemperatureImpl
+ metricUnits : boolean {declaration}
- latestTemp : float = 0.0
- driverPort : uint8_t
- calculateTemp(voltage : float) : float
TempDriver
HealthHandlerImpl
- printRoomHealth() : void
stdio
<<interface>>
Humidity
+ create()
+ meassure() : void
+ getHumidity() : uint8_t
Main
+ metricUnits : boolean = true {definition}
+ main() : int
- setupApplication() : void
- runApplication() : void
- delayMs(ms : int) : void
HumidityImpl
- lastestHumidity : uint8_t = 0
HumidityDriver

From Design to C – Part II

In Part II we will look at how we can make it more Object Oriented

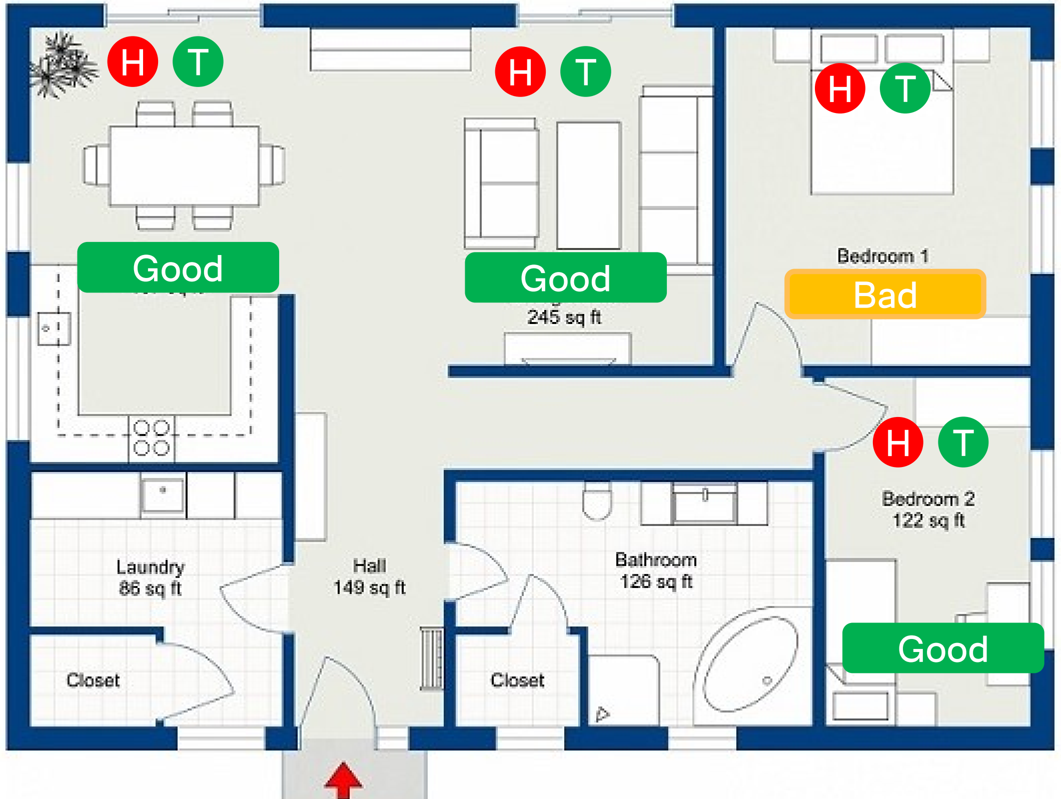
We change the design to:

The following design is for a small application that should measure and show the environment (health) in a building ***with several rooms (up to 10)***

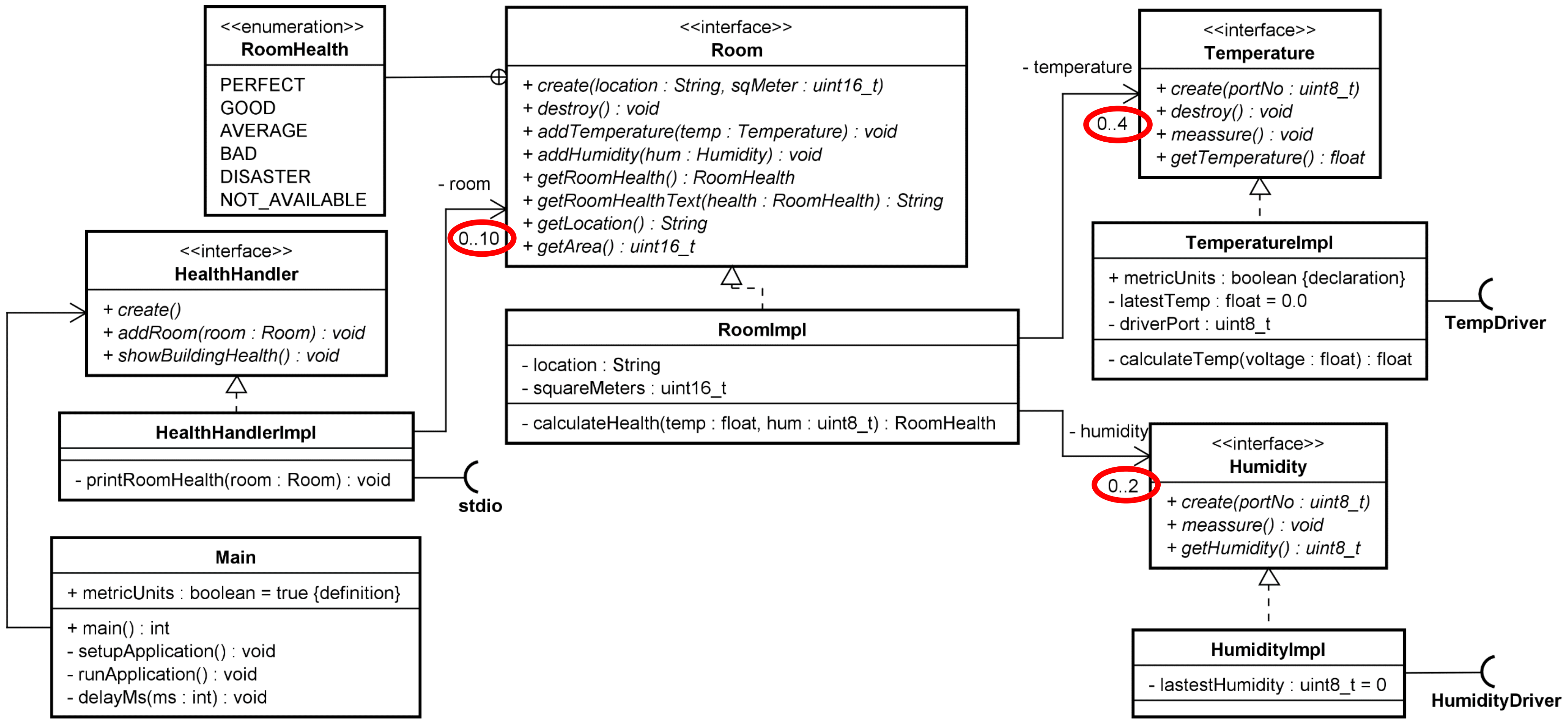
The health in a room is based on ***multiple temperature- (up to 4) and a humidity- (up to 2)*** *measuring's in the room*

The health should be shown once per second

The sensors are simulated in this example



An Example With Classes



<<enumeration>>
RoomHealth
PERFECT
GOOD
AVERAGE
BAD
DISASTER
NOT_AVAILABLE
<<interface>>
Room
+ create(location : String, sqMeter : uint16_t)
+ destroy() : void
+ addTemperature(temp : Temperature) : void
+ addHumidity(hum : Humidity) : void
+ getRoomHealth() : RoomHealth
+ getRoomHealthText(health : RoomHealth) : String
+ getLocation() : String
+ getArea() : uint16_t
- room
0..10
<<interface>>
HealthHandler
+ create()
+ addRoom(room : Room) : void
+ showBuildingHealth() : void
HealthHandlerImpl
- printRoomHealth(room : Room) : void
stdio
Main
+ metricUnits : boolean = true {definition}
+ main() : int
- setupApplication() : void
- runApplication() : void
- delayMs(ms : int) : void
RoomImpl
- location : String
- squareMeters : uint16_t
- calculateHealth(temp : float, hum : uint8_t) : RoomHealth
- temperature
0..4
<<interface>>
Temperature
+ create(portNo : uint8_t)
+ destroy() : void
+ meassure() : void
+ getTemperature() : float
TemperatureImpl
+ metricUnits : boolean {declaration}
- latestTemp : float = 0.0
- driverPort : uint8_t
- calculateTemp(voltage : float) : float
TempDriver
- humidity
0..2
<<interface>>
Humidity
+ create(portNo : uint8_t)
+ meassure() : void
+ getHumidity() : uint8_t
HumidityImpl
- lastestHumidity : uint8_t = 0
HumidityDriver

How can this be implemented in C?

C is **not** object oriented!

No – but we can do it using **Abstract Data Types (ADT)** in C

What is an Abstract Data Type in C

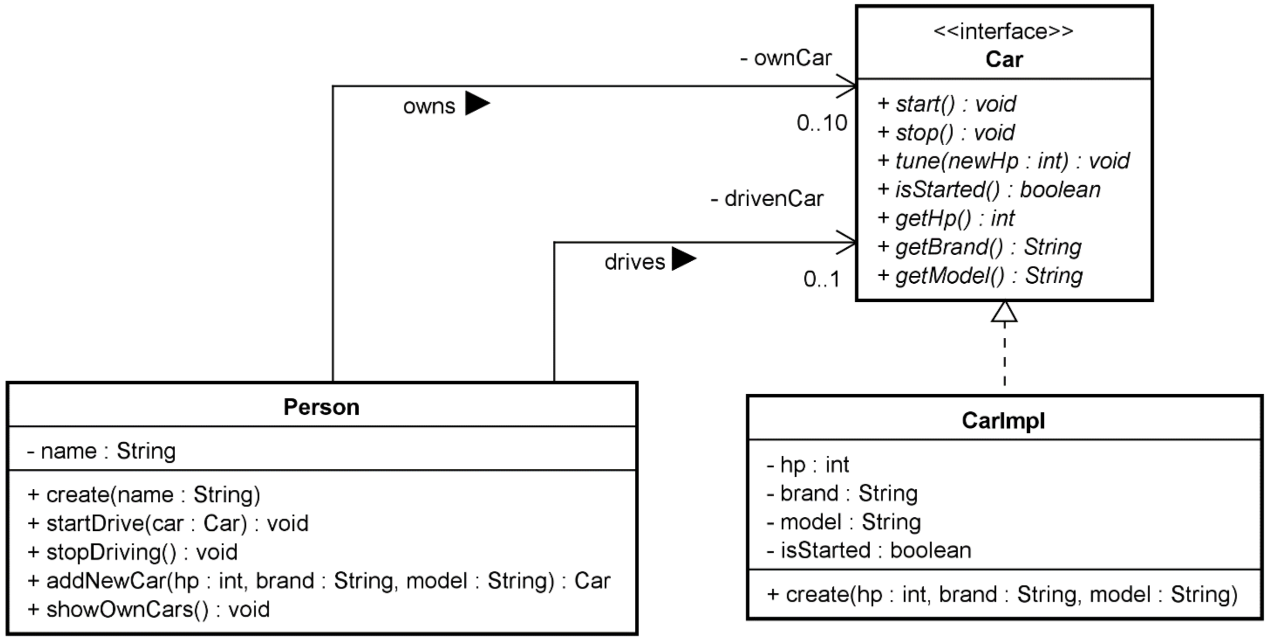
It is an **anonymous pointer** that points to a "hidden/private" struct variable

```
typedef struct car* car_t;  // The Abstract Data Type (ADT)
```
Explanation comes later ☺

The **anonymous pointer** is declared in a header file (.h)

The struct it self is defined static ("private") in the corresponding source file (.c)

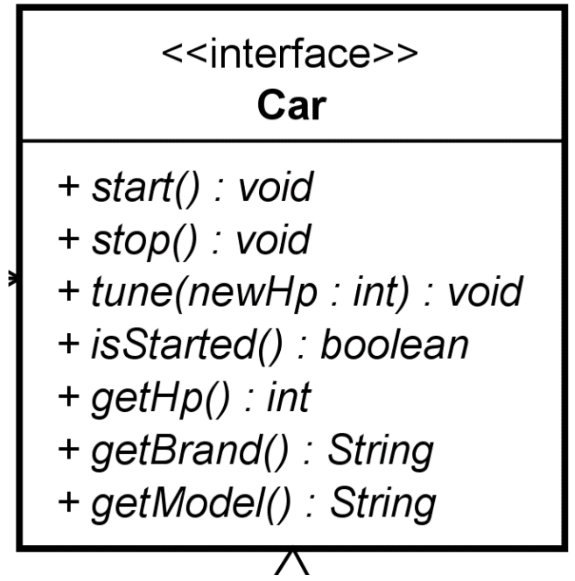
Let's look at a simple example that we implementation in both Java and C
To make the implementation comparable



Java

Car.java

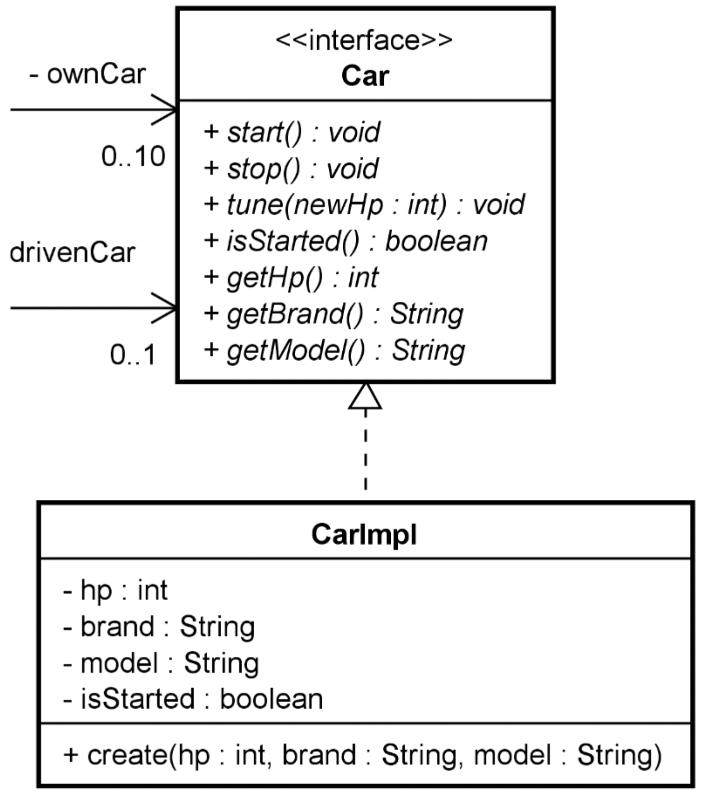
```java
public interface Car {
    public void tune(int newHp);
    public void start();
    public void stop();
    public boolean isStarted();
    public int getHp();
    public String getBrand();
    public String getModel();
}
```



Java

```
package com.via.esw1.simple_adt_example;

public class CarImpl implements Car{
    private int hp;
    private String brand;
    private String model;
    private boolean isStarted;

    public CarImpl(int hp, String brand, String model) {
        this.hp = hp;
        this.brand = brand;
        this.model = model;
        this.isStarted = false;
    }

    public void tune(int newHp) {
        hp = newHp;
    }
...
```

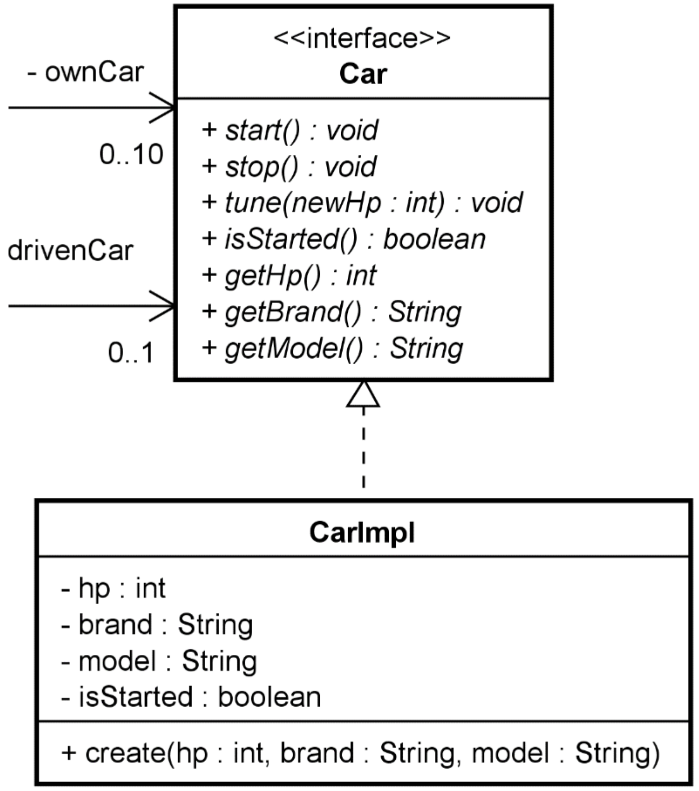
CarImpl.java



Java

```
package com.via.esw1.simple_adt_example;

public class CarImpl implements Car{
…
 public void start() {
        isStarted = true;
    }

    public void stop() {
        isStarted = false;
    }

    public boolean isStarted() {
        return isStarted;
    }

    public int getHp() {
        return hp;
    }
…
```

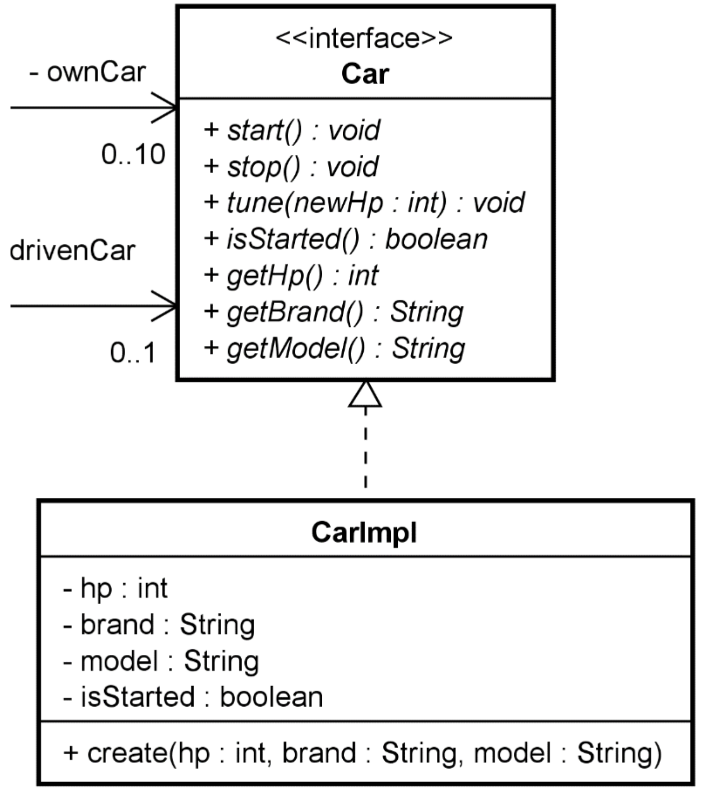
CarImpl.java



Java

CarImpl.java

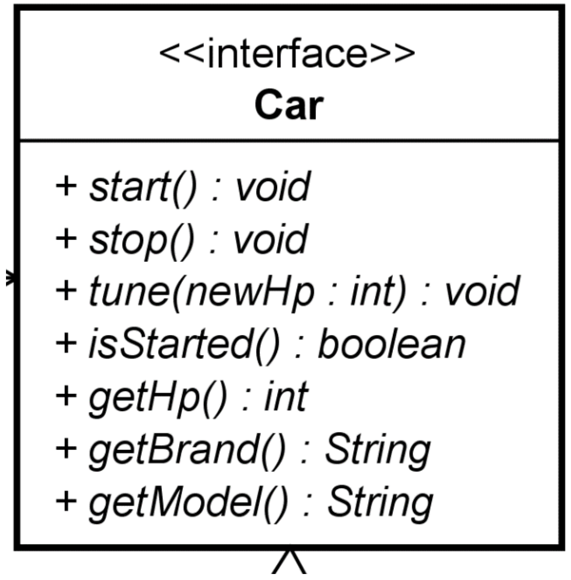
```
package com.via.esw1.simple_adt_example;

public class CarImpl implements Car{
…
 public String getBrand() {
        return brand;
    }

    public String getModel() {
        return model;
    }
}
```



C

car.h

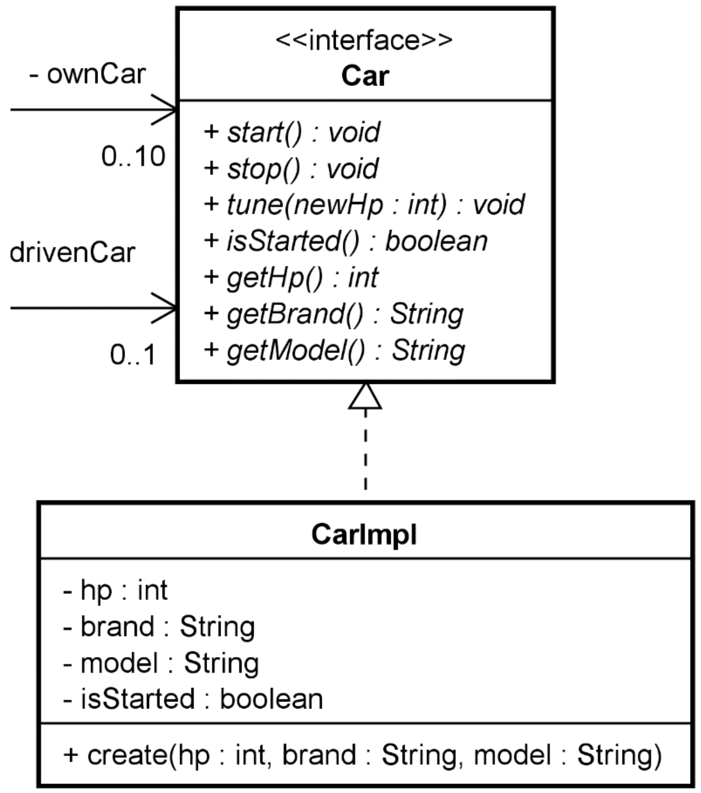
```c
#pragma once
#include <stdbool.h>

typedef struct car* car_t;  // The Abstract Data Type (ADT)

car_t car_create(int hp, char* brand, char* model);
void car_destroy(car_t self);
void car_start(car_t self);
void car_stop(car_t self);
void car_tune(car_t self, int newHp);
bool car_isStarted(car_t self);
int car_getHp(car_t self);
char* car_getBrand(car_t self);
char* car_getModel(car_t self);
```



C

car.c

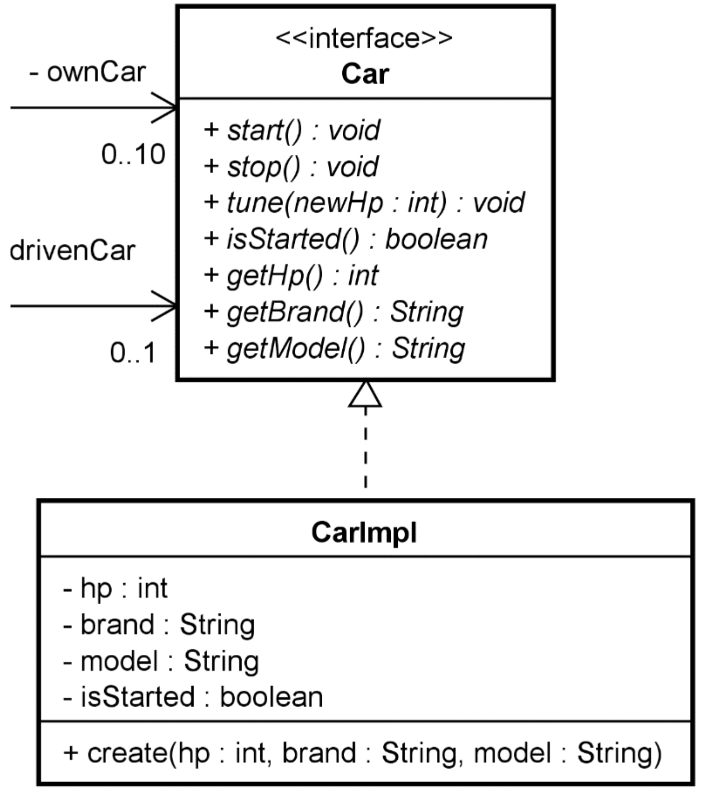
```c
#include "car.h"
#include <stdlib.h>
#include <string.h>

// The "private" fields for each "object" are defined here
typedef struct car{
    int hp;
    char brand[30];
    char model[30];
    bool isStarted;
} car;
```



C

```
/* "Constructor" ------------------------------------------ */
car_t car_create(int hp, char* brand, char* model) { // Constructor
    car_t _newCar = calloc(sizeof(car), 1);

    if (NULL == _newCar) { // There was not enough memory
      return NULL;
    }

    _newCar->hp = hp;
    strncpy(_newCar->brand, brand, sizeof(_newCar->brand) - 1);
    strncpy(_newCar->model, model, sizeof(_newCar->model) - 1);

    return _newCar;
}
```

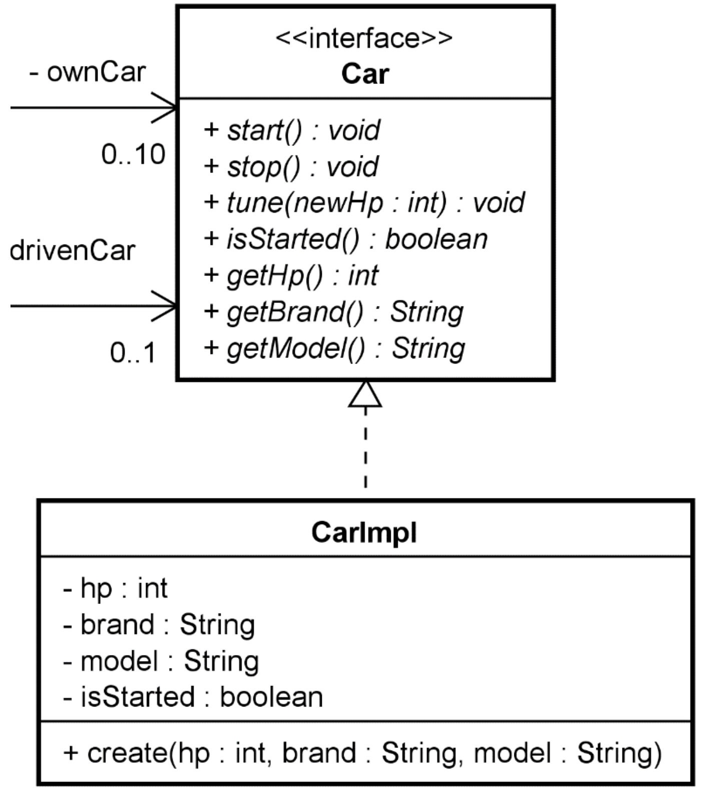
car.c



C

car.c

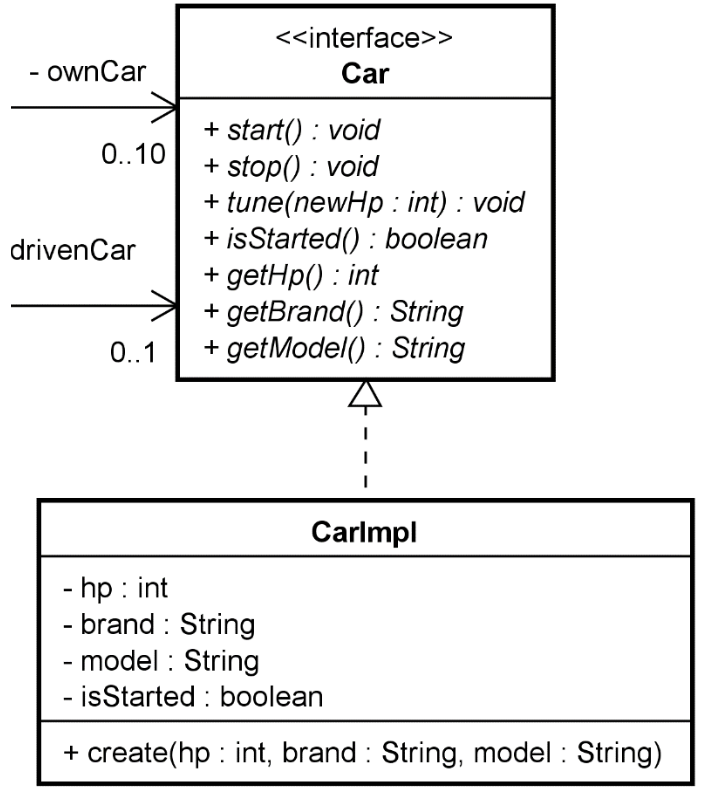
```
/* -------------------------------------------------------------
 // We need this in C - No garbage collector!!
void car_destroy(car_t self) {
   if (NULL != self) {
       free(self);
    }
}
```



C

car.c

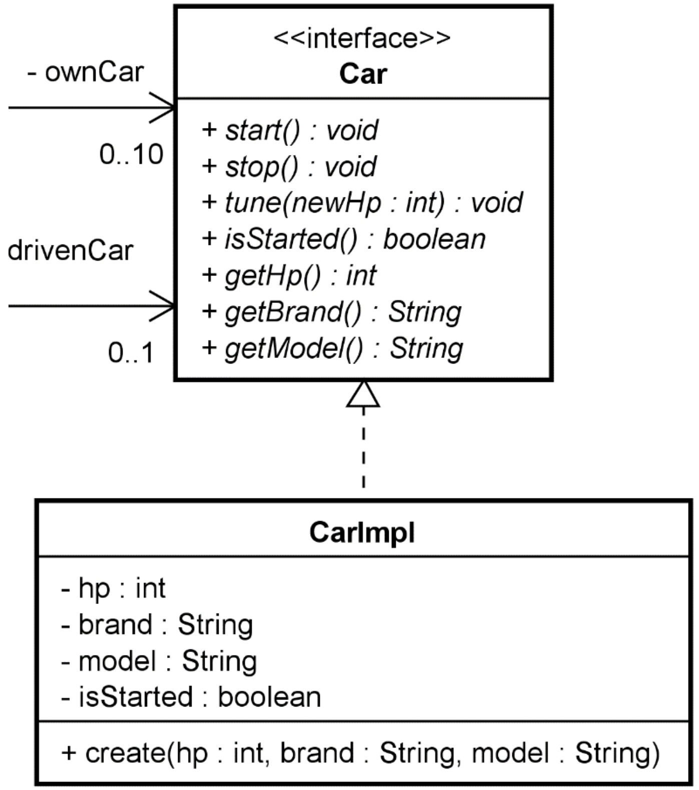
```
/* -------------------------------------------------------------------- */
void car_start(car_t self) {
    self->isStarted = true;
}

/* --------------------------------------------------------------------
void car_stop(car_t self) {
   self->isStarted = false;
}

/* --------------------------------------------------------------------
void car_tune(car_t self, int newHp) {
   self->hp = newHp;
}
```



C

car.c

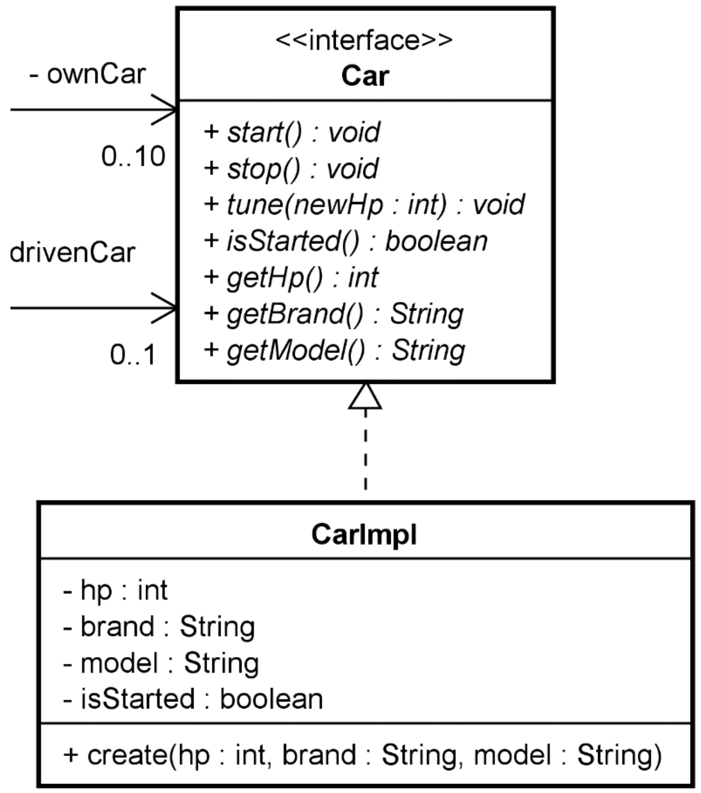
```c
/* ------------------------------------------------------------ */
bool car_isStarted(car_t self) {
   return self->isStarted;
}

/* ------------------------------------------------------------
int car_getHp(car_t self)
{
   return self->hp;
}

/* ------------------------------------------------------------
char* car_getBrand(car_t self)
{
   return self->brand;
}
```



C

car.c

```
/* -----------------------------------------------------------------------
char* car_getModel(car_t self)
{
    return self->model;
}
```



How to use the Car class in Java and C

Java

```java
Car myCar;
```

A reference to an object!

Create a new object and call constructor

```java
myCar = new CarImpl(195, "Volvo", "V60");
```

```java
myCar.start();
myCar.stop();
```

Use the reference to call methods on the object

```java
myCar = null;
```

Delete the object (garbage collector)

C

```c
car_t myCar;
```

A pointer to an “object”!

Create a new “object” and call constructor

```c
myCar = car_create(195, "Volvo", "V60");
```

```c
car_start(myCar);
car_stop(myCar);
```

Call a functions and give it the pointer to the “object”

```c
car_destroy(myCar);
```

Delete the “object” by calling the destroy function on it

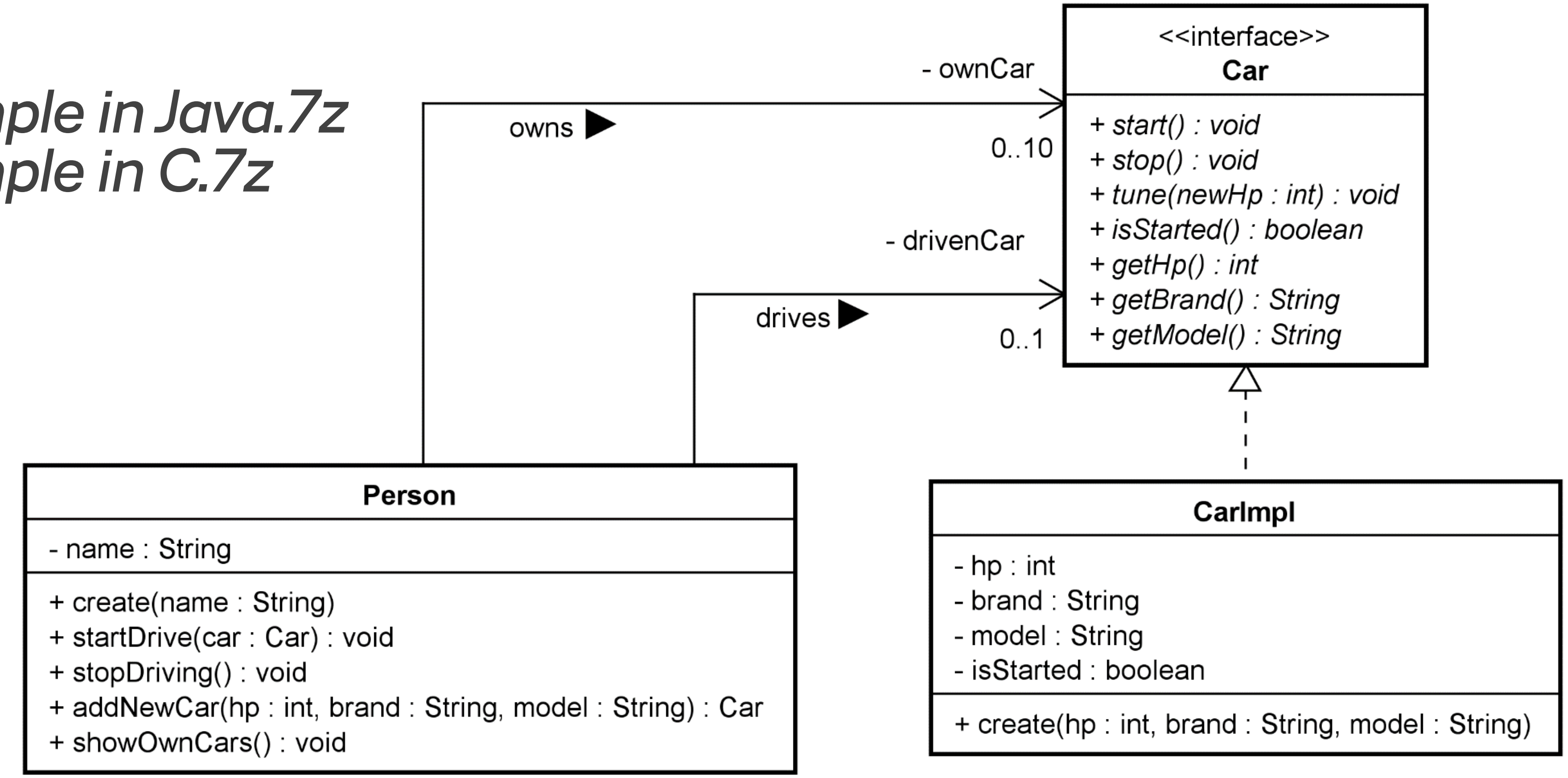
Conclusion ADT – OOP

- We can implement a kind of classes in C using ADT
- The use of our “classes” in done in a similar way as in Java/C# etc.
- We have not implemented Polymorphism, Inheritances etc.
  - **But this is of course also possible to do (out of scope for this course)**

Complete Implementation

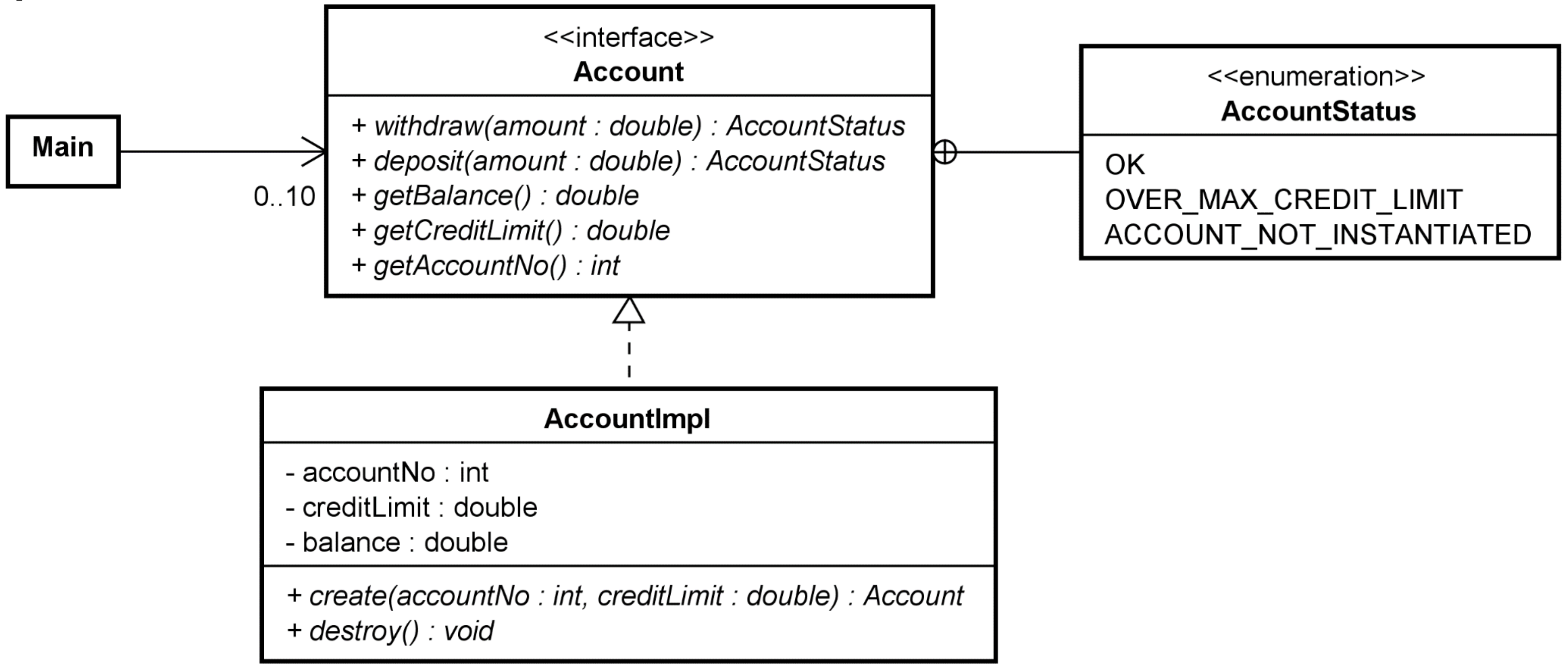
You can find my complete implementation of the design both in Java and in C in ItsLearning

Simple ADT Example in Java.7z
Simple ADT Example in C.7z



Exercise

Implement the following Bank Account as a Abstract Data Type.
Implement a small main program that instantiates 10 accounts and test its functionality



Main
0..10
<<interface>>
Account
+ withdraw(amount : double) : AccountStatus
+ deposit(amount : double) : AccountStatus
+ getBalance() : double
+ getCreditLimit() : double
+ getAccountNo() : int
<<enumeration>>
AccountStatus
OK
OVER_MAX_CREDIT_LIMIT
ACCOUNT_NOT_INSTANTIATED
AccountImpl
- accountNo : int
- creditLimit : double
- balance : double
+ create(accountNo : int, creditLimit : double) : Account
+ destroy() : void