FH-OÖ Hagenberg/HSD
SDP3, WS 2019
Übung 5

Name(1): Daniel Weyrer — Abgabetermin: 10.12.2019

Name(2): Viktoria Streibl — Punkte:

Übungsgruppe: 1 — korrigiert:

Geschätzter Aufwand in Ph: 8 | 4 — Effektiver Aufwand in Ph: 5 | 4

Beispiel 1 (24 Punkte) Fahrsimulation: Entwerfen Sie aus der nachfolgenden Spezifikation ein Klassendiagramm, instanzieren Sie dieses und implementieren Sie die Funktionalität entsprechend:

Für einen KFZ-Prüfstand ist eine Software zu entwickeln. Dabei sollen Sensorwerte zur Ermittlung der Raddrehzahlen eingelesen und verarbeitet werden. Die Aufbereitung der Daten umfasst die Berechnung der Momentangeschwindigkeit (Tachometeranzeige) und der zurückgelegten Wegstrecke (Kilometerzähler; grobe Näherung genügt).

Modellieren Sie nun einen PKW als Konkretisierung eines KFZ. Der PKW hat einen Tachometer und einen Kilometerzähler. Weiters verfügt der PKW über einen Raddrehsensor, der bei jedem Abruf die momentane Raddrehzahl eines der vier Räder zurückliefert. Bei jedem Abruf der Raddrehzahl sollen die Anzeigen für den Tachometer und die Kilometeranzeige automatisch aktualisiert werden. Verwenden Sie dazu ein passendes Design Pattern.

Der Raddrehsensor wird hier in der Simulation durch eine Datei nachgebildet. Jede Zeile der Datei entspricht dabei einer momentanen Drehzahl. Kapseln Sie den Sensor in einer Klasse. In der Schnittstelle dieser Klasse befindet sich eine Methode `GetRevolutions()`, die einen Drehzahlwert von der Datei einliest und zurückgibt.

Die Klasse PKW verfügt über eine Methode `Process()`, die vom Testtreiber fix im 500ms- Takt aufgerufen wird. `Process()` liest jedes Mal einen neuen Drehzahlwert via `GetRevolutions()` ein und legt diesen Wert lokal in einem Member ab. Weiters ist eine Methode `GetCurrentSpeed()` vorzusehen, die anhand folgender technischer Daten die Momentangeschwindigkeit ermittelt:

Einheit der Raddrehzahl: U/min
Raddurchmesser: 600mm

Das Fahrzeug verfügt über ABS und ASR (keine groben Schlupfwerte). Ermitteln Sie die Momentangeschwindigkeit entweder in m/s oder km/h.

Bsp: 500 U/min
v = (500 / 60) x 0,6m x PI x 3.6 = 56,549 km/h

Die Tachometer- und Kilometeranzeige holen sich im Zuge einer Benachrichtigung den aktuellen Geschwindigkeitswert. Die Kilometeranzeige berechnet auf Basis des 500ms-Taktes die gefahrene Wegstrecke:

(56,549 / 3,6) / 2 = 7,854m (grobe Näherung)

Simulieren Sie den Tachometer mit Hilfe der analogen Anzeige (AnalogDisplay) und die Kilometeranzeige mit Hilfe der digitalen Anzeige (DigitalDisplay).

Schreiben Sie einen Testtreiber, der wiederholt im 500ms-Takt die Methode `Process()` vom PKW aufruft.

Die Displays müssen nicht mitabgegeben werden. Die mitgelieferten Klassen für die Ansteuerung der Displays sind elektronisch mitabzugeben, allerdings kann auf einen Ausdruck im pdf verzichtet werden!

Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese. Verfassen Sie weiters eine Systemdokumentation (Funktionalität, Klassendiagramm, Schnittstellen der beteiligten Klassen, etc.)!

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!

SDP - Exercise 05

winter semester 2019/20

Viktoria Streibl - S1810306013
Daniel Weyrer - S1820306044

December 10, 2019

Contents

1 Organizational

1.1 Team

- Viktoria Streibl - S1810306013
- Daniel Weyrer - S1820306044

1.2 Roles and responsibilities

1.2.1 Jointly

- Planning
- Documentation
- Systemdocumentation
- Class Diagram

1.2.2 Viktoria Streibl

- Class IDisplay
  - Class Speed
- Testdriver

1.2.3 Daniel Weyrer

- Class Vehicle
  - Class Car
  - Class RPMSensor
- Class Distance

1.3 Effort

1.3.1 Viktoria Streibl

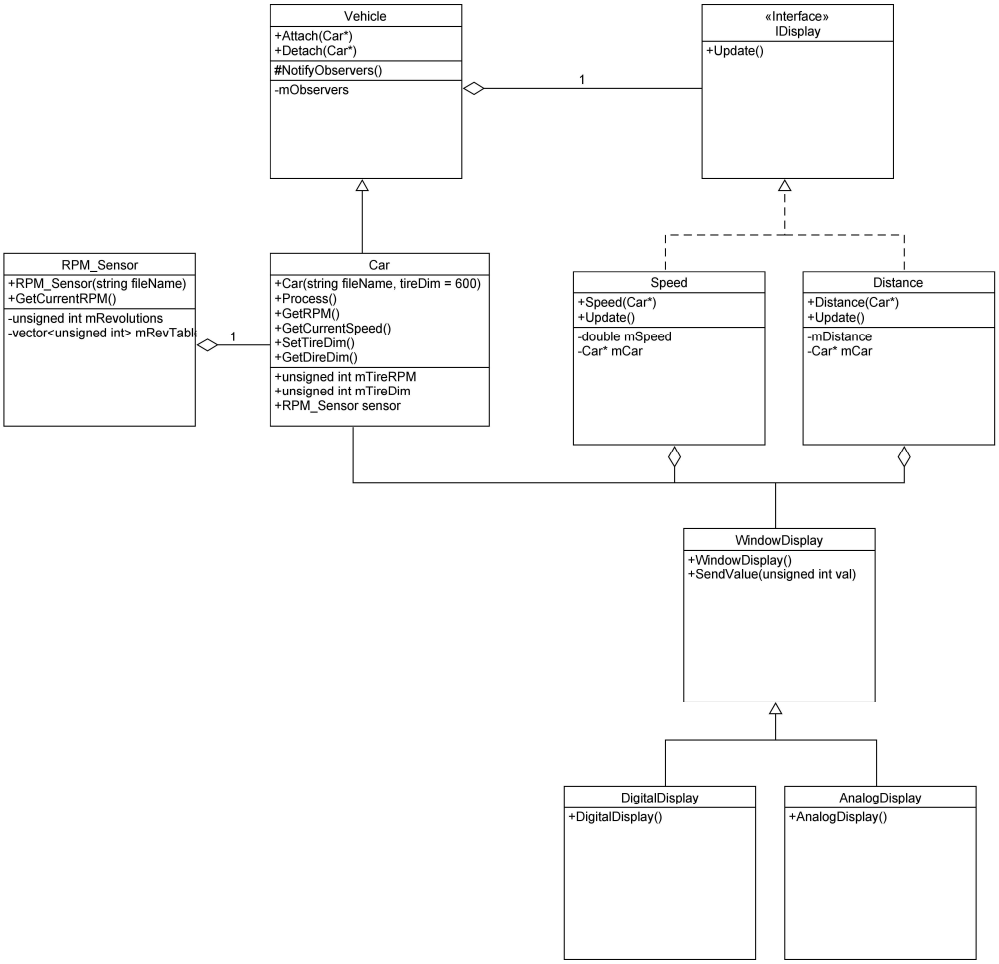
- estimated: 6 ph
- actually: 4 ph

1.3.2 Daniel Weyrer

- estimated: 8 ph
- actually: 5 ph

2 System Design

2.1 Classdiagram



2.2 Design Decisions

2.2.1 FileName in Constructor for Car

As the Sensor is only simulated and useless without a fileName, we decided to store the fileName in the car class.

2.2.2 ReadFile

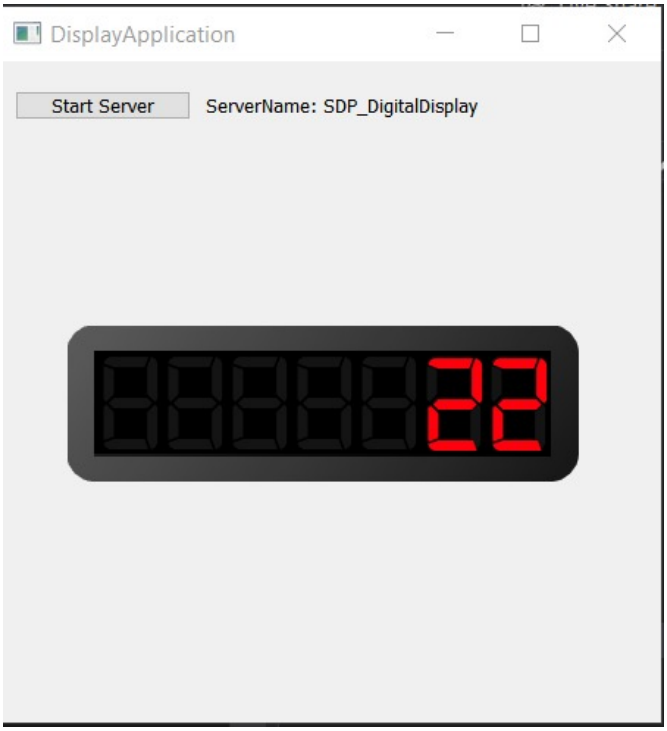
ReadFile uses the static filename given in the header to read the content to the container.

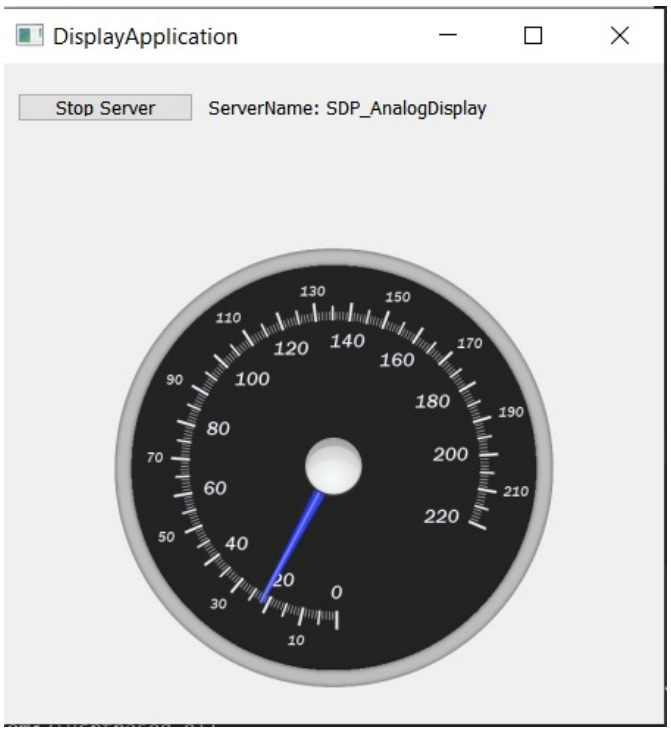
3 Test Protocol

3.1 Testfiles

3.1.1 output.txt

```
Distance: 0
Speed: 0
Distance: 0.314159
Speed: 2.26195
Distance: 1.09956
Speed: 5.65487
Distance: 2.35619
Speed: 9.04779
Distance: 4.08407
Speed: 12.4407
Distance: 6.44026
Speed: 16.9646
Distance: 9.2677
Speed: 20.3575
Distance: 13.1947
Speed: 28.2743
Distance: 17.5144
Speed: 31.1018
Distance: 22.2268
Speed: 33.9292
Distance: 26.1538
Speed: 28.2743
Distance: 30.552
Speed: 31.6673
Distance: 34.479
Speed: 28.2743
Distance: 38.0918
Speed: 26.0124
Distance: 41.2334
Speed: 22.6195
Distance: 44.0608
Speed: 20.3575
Distance: 47.3595
Speed: 23.7504
Distance: 51.6007
Speed: 30.5363
Distance: 55.3706
Speed: 27.1434
Distance: 58.5122
Speed: 22.6195
Distance: 60.8684
Speed: 16.9646
Distance: 63.6958
Speed: 20.3575
Distance: 66.3661
Speed: 19.2265
Distance: 69.3507
Speed: 21.4885
Distance: 72.4137
Speed: 22.054
```


DisplayApplication
Start Server
ServerName: SDP_DigitalDisplay
22


DisplayApplication
Stop Server
ServerName: SDP_AnalogDisplay
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220

4 Source Code

4.1 Object.h

4.1.1 header.h

```
1  /* ------------------------------------------------------------------
2  | Workfile : Object .h
3  | Description : [ HEADER ] Class for the global Object
4  | Name : Viktoria Streibl     PKZ : S1810306013
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ------------------------------------------------------------------ */
9  #ifndef OBJECT_INCLUDED
10 #define OBJECT_INCLUDED
11 class Object {
12 public:
13   // ------------------------------------------------------------------------
14   // Destructor of the class Object
15   // ------------------------------------------------------------------------
16   virtual ~Object() = default;
17 protected:
18   // ------------------------------------------------------------------------
19   // Constructor of the class Object
20   // No object of the class can be created directly
21   // ------------------------------------------------------------------------
22   Object() = default;
23 };
24 #endif //OBJECT_INCLUDED
```

4.2 Vehicle

4.2.1 Vehicle.h

```
/* ----------------------------------------------------------------------
| Workfile : Vehicle.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef VEHICLE_H
#define VEHICLE_H

#include "Object.h"
#include "IDisplay.h"

#include <vector>
#include <memory>
#include <algorithm>
#include <iostream>

class Vehicle :
  public Object {
public:

  /**
   * @brief Adds given shared_ptr to observer container
   * @param obs shared pointer of type IDisplay, which will be added to the container
   * @return
   */
  void Attach(IDisplay::SPter const& obs);

  /**
   * @brief Removes given observer-ptr from container if it‘s contained
   * @param obs shared pointer of type IDisplay, which will be added to the container
   * @return -
   */
  void Detach(IDisplay::SPter const& obs);

protected:
  /**
   * @brief iterates through vector of Observers and calls Update() on every observer
   * @param -
   * @return -
   */
  void NotifyObservers();

private:
  std::vector<IDisplay::SPter> mObservers;
};

#endif //VEHICLE_H
```

4.2.2 Vehicle.cpp

```
/* ----------------------------------------------------------------------
| Workfile : Vehicle.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#include "Vehicle.h"

void Vehicle::Attach(IDisplay::SPter const& obs) {
  try {
    if (std::find(mObservers.cbegin(), mObservers.cend(), obs) == mObservers.cend()) {
```

```
15        mObservers.emplace_back(obs);
16      }
17      else {
18        throw std::exception("Observer to attach is already contained!");
19      }
20    }
21    catch (std::exception const& ex) {
22      std::cerr << "Error while attaching: " << ex.what() << std::endl;
23    }
24    catch (std::bad_alloc const& ex) {
25      std::cerr << "Memory allocation error: " << ex.what() << std::endl;
26    }
27  }
28
29  void Vehicle::Detach(IDisplay::SPter const& obs) {
30    try {
31      auto iterObs = std::find(mObservers.cbegin(), mObservers.cend(), obs);
32      if (iterObs != mObservers.cend()) {
33        mObservers.erase(iterObs);
34      }
35      else {
36        throw std::exception("Observer to detach could not be found!");
37      }
38    }
39    catch (std::exception const& ex) {
40      std::cerr << "Error while detaching: " << ex.what() << std::endl;
41    }
42    catch (std::bad_alloc const& ex) {
43      std::cerr << "Memory allocation error: " << ex.what() << std::endl;
44    }
45
46  }
47
48  void Vehicle::NotifyObservers() {
49    for (auto elem : mObservers) {
50      elem->Update();
51    }
52  }
```

4.3 Car

4.3.1 Car.h

```
/* ------------------------------------------------------------------------
| Workfile : Car.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------------ */

#ifndef CAR_H
#define CAR_H
#include "Vehicle.h"
#include "RPMSensor.h"

#include <iostream>
class Car :
  public Vehicle {

public:

    /**
    * @brief Constructor, saves the fileName into member and initializes tireDimension and current RPMs
    * @param -
    * @return -
    */
  Car(double tireDim = 0.6) : mTireDim{ tireDim }, mTireRPM{ 0 }{}

  /**
    * @brief Calls GetRevolutions of RPM_Sensor and stores delivered value in membervariable, starts Notificationprocess
    * @param -
    * @return -
    */
  void Process();

  /**
    * @brief Getter
    * @param -
    * @return current Rotations per Minute for one Wheel
    */
  unsigned int GetRPM() const;

  /**
    * @brief Calculates current speed based on Wheel diameter and RPMs
    * @param -
    * @return current Speed
    */
  double GetCurrentSpeed() const;

  /**
    * @brief Replaces old value for Tire Dimension with the given Diameter
    * @param newDim new Dimension for Tire Diameter
    * @return -
    */
  void SetTireDim(double newDim);

  /**
    * @brief Getter
    * @param -
    * @return current Tire Diameter
    */
  double GetTireDim() const;

  using SPter = std::shared_ptr<Car>;


private:
```

```
67   RPM_Sensor mFrontRight;
68
69   unsigned int mTireRPM;
70   double mTireDim;
71
72
73 };
74
75 #endif //CAR_H
```

4.3.2 Car.cpp

```
1  /* ----------------------------------------------------------------------
2  | Workfile : Car.cpp
3  | Description : [ SOURCE ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Date : 09.12.2019
6  | Remarks : -
7  | Revision : 0
8  | ---------------------------------------------------------------------- */
9  #include "Car.h"
10
11 static const double PI = 3.14159265359;
12 static const double conversionFactorToKMH = 3.6;
13
14 void Car::Process() {
15   try {
16     mTireRPM = mFrontRight.GetRevolutions();
17   }
18   catch (std::exception const& ex) {
19     std::cerr << "Error while fetching Revolutions: "<< ex.what() << std::endl;
20   }
21   NotifyObservers();
22 }
23
24 unsigned int Car::GetRPM() const {
25   return mTireRPM;
26 }
27
28 double Car::GetCurrentSpeed() const {
29   return ((static_cast<double>(mTireRPM)/60)* mTireDim * PI * conversionFactorToKMH);
30 }
31
32 void Car::SetTireDim(double newDim) {
33   if (newDim > 0) {
34     mTireDim = newDim;
35   }
36   else {
37     throw std::exception("Tiredimension must be greater than 0!");
38   }
39 }
40
41 double Car::GetTireDim() const {
42   return mTireDim;
43 }
```

4.4 RPMSensor

4.4.1 RPMSensor.h

```
/* ------------------------------------------------------------------
| Workfile : RPM_Sensor.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */


#ifndef RPM_SENSOR_H
#define RPM_SENSOR_H

#include "Object.h"

#include <string>
#include <vector>
#include <fstream>
#include <memory>
#include <iostream>

static const std::string fileName = "testData.txt";

class RPM_Sensor : public Object {
public:
  RPM_Sensor();
  /**
   * @brief Getter
   * @param
   * @return current RPMs
   */
  unsigned int GetRevolutions();

private:
  unsigned int mRevolutions;
  std::vector<unsigned int> mRevTable;
  std::vector<unsigned int>::const_iterator currPos;


  /**
   * @brief Reads integers of file (line by line) and stores them into mRevTable
   * @param fileName with fileName
   * @return -
   */
  void ReadFile(std::string const& fileName);

};

#endif //RPM_SENSOR_H
```

4.4.2 RPMSensor.cpp

```
/* ------------------------------------------------------------------
| Workfile : RPM_Sensor.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */
#include "RPMSensor.h"


RPM_Sensor::RPM_Sensor() {
  ReadFile(fileName);
  currPos = mRevTable.cbegin();
}

```

```
17 unsigned int RPM_Sensor::GetRevolutions() {
18   if (currPos != mRevTable.cend()) {
19     mRevolutions = *currPos;
20     return (*currPos++);
21   }
22   else {
23     std::cerr<< "End of testdata reached!" <<std::endl;
24     return 0;
25   }
26 }
27
28 void RPM_Sensor::ReadFile(std::string const& fileName) {
29   unsigned int tmp;
30   //Create Filestream
31   try {
32     std::ifstream inFile{ fileName, std::ios::binary };
33
34     //Check file; throw exception in case of a fault
35     if (!inFile.good() || inFile.fail()) {
36       inFile.close();
37       throw std::exception("Error opening file!");
38     }
39
40     while (inFile >> tmp) {
41       mRevTable.push_back(tmp);
42     }
43     inFile.close();
44   }
45   catch (std::exception const& ex) {
46     std::cerr << "Exception while reading File: "<< ex.what() <<std::endl;
47   }
48   catch (std::bad_alloc const& ex) {
49     std::cerr << "Memory allocation error: " << ex.what() << std::endl;
50   }
51   catch (...) {
52     std::cerr << "Unhandled exception!" << std::endl;
53   }
54 }
```

4.5 IDisplay

4.5.1 IDisplay.h

```
1  /* ------------------------------------------------------------------------
2  | Workfile : IDisplay.h
3  | Description : [ HEADER ]
4  | Name : Daniel Weyrer      PKZ : S1820306044
5  | Date : 09.12.2019
6  | Remarks : -
7  | Revision : 0
8  | ------------------------------------------------------------------------ */
9
10 #ifndef IDISPLAY_H
11 #define IDISPLAY_H
12
13 #include <memory>
14 #include "WindowsDisplay.h"
15
16 class IDisplay {
17
18 public:
19   /**
20    * @brief pure virtual Function, ready to be implemented in an concrete observer
21    * @param -
22    * @return -
23    */
24   virtual void Update() = 0;
25
26   virtual ~IDisplay() = default;
27
28   using SPter = std::shared_ptr<IDisplay>;
29   WindowsDisplay::SPtr anaDisp = std::make_shared<AnalogDisplay>();
30   WindowsDisplay::SPtr digDisp = std::make_shared<DigitalDisplay>();
31 };
32
33 #endif //IDISPLAY_H
```

4.6 Speed

4.6.1 Speed.h

```
/* ---------------------------------------------------------------------
| Workfile : Speed.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| --------------------------------------------------------------------- */


#ifndef SPEED_H
#define SPEED_H
#include "IDisplay.h"
#include "Car.h"
class Speed : public IDisplay {
public:
  /**
   * @brief constructor for Speed-Observer
   * @param Shared-Pointer to a Car
   * @return -
   */
  Speed(Car::SPter const& car) : mCar{ car }, mSpeed{ car->GetCurrentSpeed() } {}

  /**
   * @brief gets current speed of the Car pointed to
   * @param -
   * @return -
   */
  virtual void Update() override;

private:
  double mSpeed;
  Car::SPter mCar;
};

#endif //SPEED_H
```

4.6.2 Speed.cpp

```
/* ---------------------------------------------------------------------
| Workfile : Speed.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
     Viktoria Streibl   PKZ : S1810306013
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| --------------------------------------------------------------------- */

#include "Speed.h"

void Speed::Update() {

  mSpeed = mCar->GetCurrentSpeed();

  if (digDisp->SendValue(mSpeed)) {
    std::cout << "Speed: " << mSpeed << std::endl;
  }

}
```

4.7 Distance

4.7.1 Distance.h

```
/* ------------------------------------------------------------------
| Workfile : Distance.h
| Description : [ HEADER ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */

#ifndef DISTANCE_H
#define DISTANCE_H

#include "IDisplay.h"
#include "Car.h"

class Distance : public IDisplay {
public:
  /**
   * @brief constructor for concrete observer Distance; initializes Shared-Pointer and Distance
   * @param shared pointer to a car
   * @return
   */
  Distance(Car::SPter const& car) : mCar{ car }, mDistance{ 0 } {}

  /**
   * @brief requests current revolutions, calculates distance travelled and saves it into member
   * @param -
   * @return -
   */
  virtual void Update() override;

private:
  double mDistance;
  Car::SPter mCar;

  /**
   * @brief Calculates Distance based on the current Speed
   * @param velocity: current velocity the car travels
   * @return current Distance travelled
   */
  double CalcDistance(double const velocity) const;
};

#endif //DISTANCE_H
```

4.7.2 Distance.cpp

```
/* ------------------------------------------------------------------
| Workfile : Distance.cpp
| Description : [ SOURCE ]
| Name : Daniel Weyrer      PKZ : S1820306044
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */

#include "Distance.h"

static const double conversionFactorToKMH = 3.6;

void Distance::Update() {

  mDistance += CalcDistance(mCar->GetCurrentSpeed());
  if (anaDisp->SendValue(mDistance))
  {
    std::cout << "Distance: " << mDistance << std::endl;
  }
}
```

```
22
23 double Distance::CalcDistance(double const velocity) const {
24   if (velocity != 0) {
25     return ((velocity) / conversionFactorToKMH) / 2;
26   }
27   return 0;
28 }
```

4.8 WindowsDisplay

4.8.1 WindowsDisplay.h

```
#ifndef WINDOWDISPLAY_H
#define WINDOWDISPLAY_H

#include <Windows.h>
#include <string>
#include <iostream>
#include <memory>
#include "Object.h"

enum eCommand {
   ConfigureAnalogOnly  = 0,
   ConfigureDigitalOnly = 1,
   ConfigureBoth = 2,
   SetValueAnalogOnly = 3,
   SetValueDigitalOnly = 4,
   SetValueBoth = 5
};

struct DataPacket {
   eCommand command;
   unsigned int value;
};

class WindowsDisplay : public Object {
public:
   WindowsDisplay(std::string const& pipeName);
   ~WindowsDisplay();

   bool SendValue(unsigned int value);
   typedef std::shared_ptr<WindowsDisplay> SPtr;
protected:
   WindowsDisplay(WindowsDisplay const&);
   WindowsDisplay& operator=(WindowsDisplay const&);

private:
   HANDLE mhPipe;

   void Init(std::string const& pipeName);
   bool Send(unsigned int value);
};



class DigitalDisplay: public WindowsDisplay {
public:
   DigitalDisplay(): WindowsDisplay("SDP_DigitalDisplay") {
   }
};


class AnalogDisplay: public WindowsDisplay {
public:
   AnalogDisplay(): WindowsDisplay("SDP_AnalogDisplay") {
   }
};

#endif //WINDOWDISPLAY_H
```

4.8.2 WindowsDisplay.cpp

```
#ifndef WINDOWDISPLAY_H
#define WINDOWDISPLAY_H

#include <Windows.h>
#include <string>
#include <iostream>
#include <memory>
#include "Object.h"
```

```

enum eCommand {
   ConfigureAnalogOnly  = 0,
   ConfigureDigitalOnly = 1,
   ConfigureBoth = 2,
   SetValueAnalogOnly = 3,
   SetValueDigitalOnly = 4,
   SetValueBoth = 5
};

struct DataPacket {
   eCommand command;
   unsigned int value;
};

class WindowsDisplay : public Object {
public:
   WindowsDisplay(std::string const& pipeName);
   ~WindowsDisplay();

   bool SendValue(unsigned int value);
   typedef std::shared_ptr<WindowsDisplay> SPtr;
protected:
   WindowsDisplay(WindowsDisplay const&);
   WindowsDisplay& operator=(WindowsDisplay const&);

private:
   HANDLE mhPipe;

   void Init(std::string const& pipeName);
   bool Send(unsigned int value);
};



class DigitalDisplay: public WindowsDisplay {
public:
   DigitalDisplay(): WindowsDisplay("SDP_DigitalDisplay") {
   }
};


class AnalogDisplay: public WindowsDisplay {
public:
   AnalogDisplay(): WindowsDisplay("SDP_AnalogDisplay") {
   }
};

#endif //WINDOWDISPLAY_H
```

4.9 TestDriver

4.9.1 TestDriver.cpp

```
/* ------------------------------------------------------------------
| Workfile : TestDriver.cpp
| Description : [ SOURCE ]
| Name : Viktoria Streibl     PKZ : S1810306013
| Date : 09.12.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */

#include "Car.h"
#include "IDisplay.h"
#include "Distance.h"
#include "Speed.h"
#include <windows.h>

#define LOOP_DURATION 25

int main() {
```

```
19    bool exit = false;
20
21    Car::SPter pCar{ std::make_shared<Car>() };
22    IDisplay::SPter distance{ std::make_shared<Distance>(pCar) };
23    IDisplay::SPter speed{ std::make_shared<Speed>(pCar) };
24
25    pCar->Attach(distance);
26    pCar->Attach(speed);
27
28    for(int i = 0; i < LOOP_DURATION; i++){
29      pCar->Process();
30      Sleep(500);
31    }
32
33    return 0;
34  }
```

5 Methoddescriptions

Chapter 3

Class Documentation

3.1 AnalogDisplay Class Reference

Inherits WindowsDisplay.

Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/WindowsDisplay.h

3.2 Car Class Reference

Inherits Vehicle.

Public Types

- using **SPter** = std::shared_ptr< Car >

Public Member Functions

- Car (double tireDim=0.6)

  Constructor, saves the fileName into member and initializes tireDimension and current RPMs.
- void Process ()

  Calls GetRevolutions of RPM_Sensor and stores delivered value in membervariable, starts Notificationprocess.
- unsigned int GetRPM () const

  Getter.
- double GetCurrentSpeed () const

  Calculates current speed based on Wheel diameter and RPMs.
- void SetTireDim (double newDim)

  Replaces old value for Tire Dimension with the given Diameter.
- double GetTireDim () const

  Getter.

Additional Inherited Members

3.2.1 Constructor & Destructor Documentation

3.2.1.1 Car()

```
Car::Car (
            double tireDim = 0.6 )  [inline]
```

Constructor, saves the fileName into member and initializes tireDimension and current RPMs.

Parameters

-	

Returns

-

3.2.2 Member Function Documentation

3.2.2.1 GetCurrentSpeed()

```
double Car::GetCurrentSpeed ( ) const
```

Calculates current speed based on Wheel diameter and RPMs.

Parameters

-	

Returns

current Speed

3.2.2.2 GetRPM()

```
unsigned int Car::GetRPM ( ) const
```

Getter.

Parameters

-	

Returns

current Rotations per Minute for one Wheel

3.2.2.3 GetTireDim()

```
double Car::GetTireDim ( ) const
```

Getter.

Parameters

-	

Returns

current Tire Diameter

3.2.2.4 Process()

```
void Car::Process ( )
```

Calls GetRevolutions of RPM_Sensor and stores delivered value in membervariable, starts Notificationprocess.

Parameters

-	

Returns

-

3.2.2.5 SetTireDim()

```
void Car::SetTireDim (
            double newDim )
```

Replaces old value for Tire Dimension with the given Diameter.

Parameters

newDim	new Dimension for Tire Diameter

Returns

-

The documentation for this class was generated from the following files:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Car.h
- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Car.cpp

3.3 DataPacket Struct Reference

Public Attributes

- eCommand **command**
- unsigned int **value**

The documentation for this struct was generated from the following file:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/WindowsDisplay.h

3.4 DigitalDisplay Class Reference

Inherits WindowsDisplay.

Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/WindowsDisplay.h

3.5 Distance Class Reference

Inherits IDisplay.

Public Member Functions

- Distance (Car::SPter const &car)

  constructor for concrete observer Distance; initializes Shared-Pointer and Distance
- virtual void Update () override

  requests current revolutions, calculates distance travelled and saves it into member

Additional Inherited Members

3.5.1 Constructor & Destructor Documentation

3.5.1.1 Distance()

```
Distance::Distance (
            Car::SPter const & car )  [inline]
```

constructor for concrete observer Distance; initializes Shared-Pointer and Distance

Parameters

shared	pointer to a car

Returns

3.5.2 Member Function Documentation

3.5.2.1 Update()

```
void Distance::Update ( )  [override], [virtual]
```

requests current revolutions, calculates distance travelled and saves it into member

Parameters

-	

Returns

-

Implements IDisplay.

The documentation for this class was generated from the following files:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Distance.h
- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Distance.cpp

3.6 IDisplay Class Reference

Inherited by Distance, and Speed.

Public Types

- using **SPter** = std::shared_ptr< IDisplay >

Public Member Functions

- virtual void Update ()=0

  pure virtual Function, ready to be implemented in an concrete observer

Public Attributes

- WindowsDisplay::SPtr **anaDisp** = std::make_shared<AnalogDisplay>()
- WindowsDisplay::SPtr **digDisp** = std::make_shared<DigitalDisplay>()

3.6.1 Member Function Documentation

3.6.1.1 Update()

```
virtual void IDisplay::Update ( )  [pure virtual]
```

pure virtual Function, ready to be implemented in an concrete observer

Parameters

-	

Returns

-

Implemented in Distance, and Speed.

The documentation for this class was generated from the following file:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/IDisplay.h

3.7 Object Class Reference

Inherited by RPM_Sensor, Vehicle, and WindowsDisplay.

The documentation for this class was generated from the following file:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Object.h

3.8 RPM_Sensor Class Reference

Inherits Object.

Public Member Functions

- unsigned int GetRevolutions ()

  Getter.

3.8.1 Member Function Documentation

3.8.1.1 GetRevolutions()

```
unsigned int RPM_Sensor::GetRevolutions ( )
```

Getter.

Parameters

The documentation for this class was generated from the following files:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/RPMSensor.h
- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/RPMSensor.cpp

3.9 Speed Class Reference

Inherits IDisplay.

Public Member Functions

- Speed (Car::SPter const &car)

  constructor for Speed-Observer
- virtual void Update () override

  gets current speed of the Car pointed to

Additional Inherited Members

3.9.1 Constructor & Destructor Documentation

3.9.1.1 Speed()

```
Speed::Speed (
            Car::SPter const & car )  [inline]
```

constructor for Speed-Observer

Parameters

Shared-Pointer	to a Car

Returns

-

3.9.2 Member Function Documentation

3.9.2.1 Update()

```
void Speed::Update ( )  [override], [virtual]
```

gets current speed of the Car pointed to

Parameters

-	

Returns

-

Implements IDisplay.

The documentation for this class was generated from the following files:

- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Speed.h
- C:/Users/Daniel/OneDrive-students.fh-hagenberg.at/SS19/OneDrive - students.fh-hagenberg.at/FH/Sem 3 - 19_20/SDP3/UE/Git_Work/Exersice5/DrivingSimulation/DrivingSimulation/Speed.cpp

3.10 Vehicle Class Reference

Inherits Object.

Inherited by Car.

Public Member Functions

- void Attach (IDisplay::SPter const &obs)

  Adds given shared_ptr to observer container.
- void Detach (IDisplay::SPter const &obs)

  Removes given observer-ptr from container if it`s contained.

Protected Member Functions

- void NotifyObservers ()

  iterates through vector of Observers and calls Update() on every observer

3.10.1 Member Function Documentation

3.10.1.1 Attach()

```
void Vehicle::Attach (
            IDisplay::SPter const & obs )
```

Adds given shared_ptr to observer container.

Parameters

obs	shared pointer of type IDisplay, which will be added to the container