FH-OÖ Hagenberg/HSD
SDP3, WS 2019
Übung 8

Name(1): Daniel Weyrer — Abgabetermin: 14.01.2020

Name(2): Viktoria Streibl — Punkte:

Übungsgruppe: 1 — korrigiert:

Geschätzter Aufwand in Ph: 6 | 6 — Effektiver Aufwand in Ph: 6 | 3

Beispiel 1 (24 Punkte) Robotersteuerung: Entwerfen Sie aus der nachfolgend gegebenen Spezifikation ein Klassendiagramm, instanzieren Sie dieses, implementieren Sie die Funktionalität entsprechend und verwenden Sie dazu das Command-Pattern:

Für eine Robotersteuerung soll eine Software implementiert werden die verschiedenen Robotern (Hexapod-Roboter, Radroboter,...) unterschiedliche Kommandos geben kann.

Die Steuerung hat unter anderem folgende Schnittstelle:

```
void AddCommand(/*Command*/);   //adds a command to command list
void Start();                   //starts execution of all commands in list
void Undo(size_t const count);  //undo and removes specified count of commands
void Reset();                   //removes all commands
```

Folgende Kommandos sollen implementiert werden:

- TurnLeft: Der Roboter dreht sich um eine Vierteldrehung nach links.
- TurnRight: Der Roboter dreht sich um eine Vierteldrehung nach rechts.
- Forward: Der Roboter bewegt sich in die aktuelle Richtung.
- MacroMovement: Dem Roboter können mehrere Kommandos in einem mitgeteilt werden.

Ein Roboter speichert einen Namen, die aktuelle Position (x,y-Koordinaten) und die Richtung (Norden, Osten, Süden, Westen).

Eine `Client`-Klasse verwaltet die einzelnen Roboter und gibt deren aktuellen Zustände auf `std::cout` aus. Dazu bieten die Roboter folgende Schnittstelle an:

```
class IRobot
```

```
{
public:
   virtual void Info(std::ostream& os) const = 0;
   virtual ~IRobot();
};
```

Die Ausgabe der Roboterinformation sieht folgendermaßen aus:

```
WheelRobot: Robin, Pos(300,900), Direction(WEST)
Hexapod: Maximus, Pos(1300,900), Direction(EAST)
Hexapod: Paulo, Pos(-200,400), Direction(SOUTH)
...
```

Testen Sie ausführlich alle Funktionen und die Steuerung der Roboter. Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen. 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 08

winter semester 2019/20

Viktoria Streibl - S1810306013
Daniel Weyrer - S1820306044

January 13, 2020

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

- Client
- Hexapod
- IRobot
- Object
- Robot
- Wheelbot
- TestDriver

1.2.3 Daniel Weyrer

- Control
- Forward
- ICommand
- MacroMovement
- TestDriver
- TurnLeft
- TurnRight

1.3 Effort

1.3.1 Viktoria Streibl

- estimated: 6 ph
- actually: 3 ph

1.3.2 Daniel Weyrer

- estimated: 6 ph
- actually: 6 ph

2 Requirenment Definition(System Specification)

This program should reflect robot control. Different robots should receive different commands and react accordingly. The following commands were requested:

- Turn Left
- Turn Right
- Forward
- MacroMovement - get multiple commands

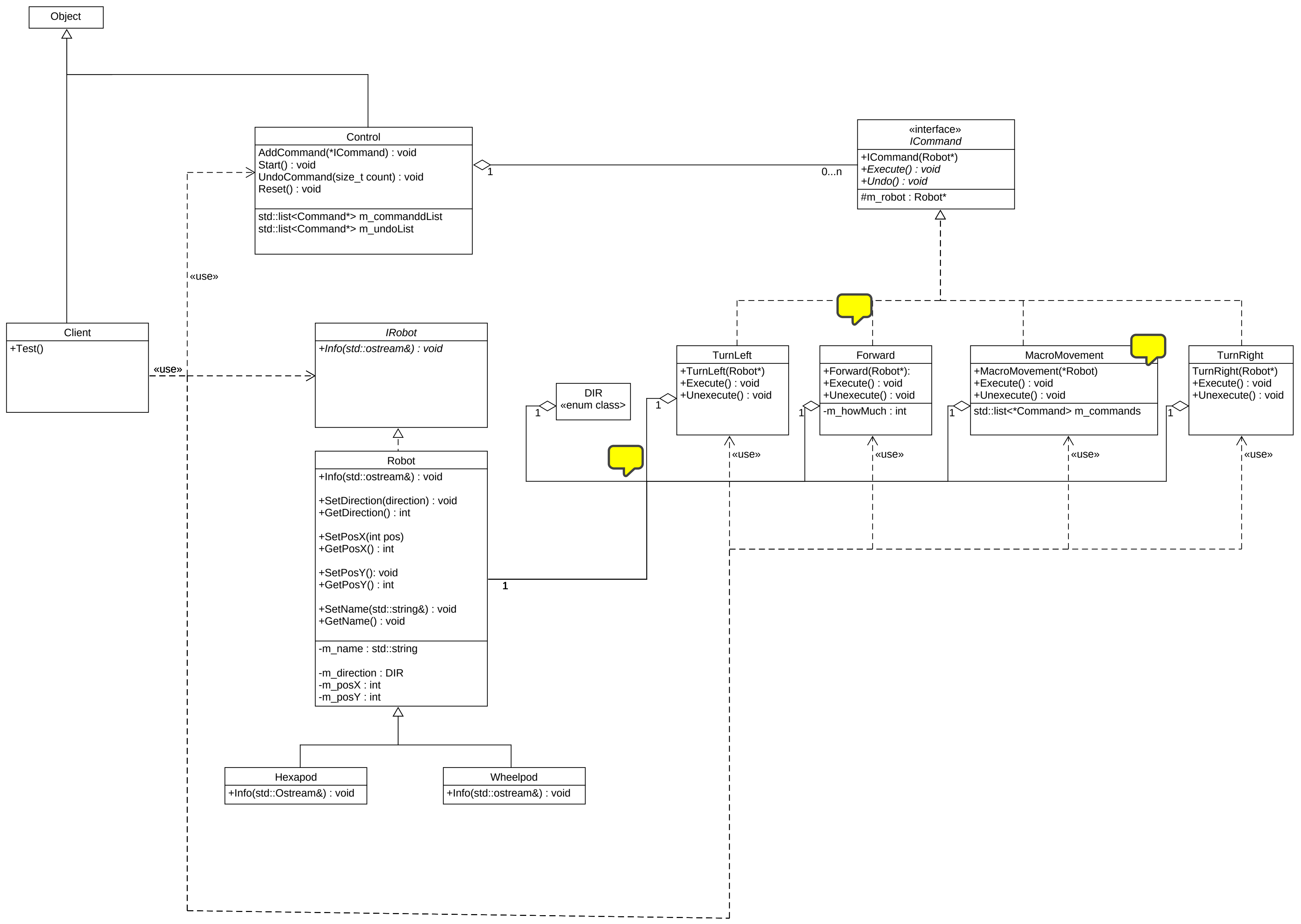
3 System Design

3.1 Design Decisions

3.1.1 Case-Statements in TurnLeft and TurnRight

To make the commands independent from each other (so there‘s no need to implement a TurnRight when there is a TurnLeft and the other way round), both Commands have the complete Implementation for their Undo-Command!

3.2 Classdiagram



4 Component Design

4.1 Client

The client class has only one function which tests the robot control system. It creates two different robots, a hexapod and a wheelbot and sends them commands. It also checks for problems with nullptr as parameter for various calls.

4.2 Control

The Control class manages the Robots by saving all Commands to be applied on them in a vector. To undo all Commands, they are being saved in the undoList after they have been executed. It has also following functions:

- Add Comment

  This function gets a command and add it to the command list

- Start

  This functions execute all of the commands ins the command list

- Undo

  This function adds a command to the undo-list

- Reset

  This function removes all added commands from the command list. The commands in the Undo-List are kept to reset the Robot if needed.

4.3 Hexapod

This module is a robot type and inherits from the class Robot. It has an constructor which has the name of the robot as parameter and sets it via cTor-Call from Robot. It also has a function Info, which prints all the informations of this robot.

4.4 Wheelbot

This module is a robot type and inherits from the class Robot. It has an constructor which has the name of the robot as parameter and sets it and sets it via cTor-Call from Robot. It also has a function Info, which prints all the informations of this robot.

4.5 ICommand

This is an Interface and stores the robot, which is given via the constructor. It declares the functions Execute() and Unexecute(). Used to create Pointers of this Type and Call both functions (Unexecute() reverts the Actions Execute() did)

It helds a shared pointer to the Robot the Command is meant for - to avoid code-duplication, the Constructor has been moved to this interface!

4.6 IRobot

IRobot is an interface. It declares the method Info with and ostream parameter to print current Location and the direction of the Robot.

4.7 Robot

This modul has an enum class DIR, it contains the directions north, south, east and west. The class Robot inherits from the Interface IRobot. It stores the name, direction, x-, and y-position. It also implements the following Get-/Setmethods:

- Set/Get-Direction

  The setter gets a direction and saves it. The getter returns the stored direction of the robot.

- Set/Get-PosX

  The setter gets the x-position and saves it. The getter returns the stored direction of the robot.

- Set/Get-PosY

  The setter gets the y-position and saves it. The getter returns the stored direction of the robot.

- Set/Get-Name

  The setter gets the name of the robot and saves it. The getter returns the stored direction of the robot.

4.8 Forward

It overrides the functions Execute and Unexecute. These functions increase or decrease the position x,y depending on the direction.

4.9 TurnLeft

Changes the direction of the Robot by 90 degrees clockwise.

4.10 TurnRight

Changes the direction of the Robot by 90 degrees counter-clockwise.

4.11 MacroMovement

Takes a vector of shared pointers of commands. When calling the function Execute(), all Commands in the vector are being Execute one by one and stored in the undo-list, to revert all the changes it did.

5 Source Code

5.1 Client

5.1.1 Client.h

```cpp
/* -----------------------------------------------------------------------
| Workfile : Client.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ----------------------------------------------------------------------- */

#ifndef CLIENT_H
#define CLIENT_H

#include "vld.h"

#include "Control.h"
#include "Forward.h"
#include "TurnLeft.h"
#include "TurnRight.h"
#include "MacroMovement.h"


#include "Robot.h"
#include "Wheelbot.h"
#include "Hexapod.h"

#include <iostream>
#include <memory>



class Client : public Object {
public:
  void Test();
};

#endif // !CLIENT_H
```

5.1.2 Client.cpp

```cpp
#include "Client.h"


void Client::Test() {

  Robot::SPter Robert{ std::make_shared<Hexapod>("Robert") };
  IRobot::SPter IRobert{ Robert };
  Robot::SPter Harry{ std::make_shared<Wheelbot>("Harry") };
  IRobot::SPter IHarry{ Harry };

  Control ctrl;

  Control::TcmdList cmd;
  cmd.emplace_back(std::make_shared<Forward>(Harry, -300));
  cmd.emplace_back(std::make_shared<TurnLeft>(Harry));
  cmd.emplace_back(std::make_shared<Forward>(Harry, 500));
  cmd.emplace_back(nullptr);


  ctrl.AddCommand(std::make_shared<Forward>(Harry, 300));
  ctrl.AddCommand(std::make_shared<TurnLeft>(Harry));
  ctrl.AddCommand(std::make_shared<TurnRight>(Robert));
  ctrl.AddCommand(std::make_shared<Forward>(Robert, 400));
  ctrl.AddCommand(std::make_shared<MacroMovement>(Harry, cmd));
  ctrl.AddCommand(nullptr);



  ctrl.Start();

  IRobert->Info(std::cout);
  IHarry->Info(std::cout);

  ctrl.Undo(6);

  IRobert->Info(std::cout);
  IHarry->Info(std::cout);

}
```

5.2 Control

5.2.1 Control.h

```cpp
/* -----------------------------------------------------------------------
| Workfile : Control.h
| Description : [ HEADER ]
| Name : Daniel Weyrer    PKZ : S1820306044
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ----------------------------------------------------------------------- */

#ifndef OBJECT_H
#define OBJECT_H

#include "Object.h"
#include "ICommand.h"
#include <vector>
#include <deque>

#include <algorithm>
#include <memory>

class Control : public Object {
public:
  //typedefs
  typedef std::shared_ptr<ICommand> TcmdPtr;
  typedef std::vector<TcmdPtr> TcmdList;

  //Adds command to Commandlist
  void AddCommand(TcmdPtr cmd);

  //Starts execution of all commands in Commandlist
  void Start();

  //executes Unexecute() of "count"-Items in Undo-List
  void Undo(size_t const count);

  //Removes all added Commands
  void Reset();

private:
  TcmdList m_commands;
  TcmdList m_undoList;
};

#endif // !OBJECT_H
```

5.2.2 Control.cpp

```cpp
#include "Control.h"
#include "Control.h"

void Control::AddCommand(TcmdPtr cmd) {
  try {
    if (cmd == nullptr) {
      throw std::exception("Given Command is nullptr!");
    }
    m_commands.emplace_back(cmd);
  }
  catch (std::bad_alloc const& ex) {
    std::cerr << "Memory allocation: " << ex.what() << std::endl;
  }
  catch (std::exception const& ex) {
    std::cerr << "Exception in AddCommand: " << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unhandled exception in AddCommand()!" << std::endl;
  }
}

```

```cpp
void Control::Start() {
  try {
    //iterate through vector and call Execute() on all elements, copy to undolist after execution
    for (auto elem : m_commands) {
      elem->Execute();
      m_undoList.emplace_back(elem);
    }
  }
  catch (std::bad_alloc const& ex) {
    std::cerr << "Memory allocation: " << ex.what() << std::endl;
  }
  catch (std::exception const& ex) {
    std::cerr << "Exception during Execution: " << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unhandled exception in Start()!" << std::endl;
  }

}

void Control::Undo(size_t const count) {
  try {
    size_t i = 0;
    while(i < count && m_undoList.size() > 0 ) {
      m_undoList.back()->Unexecute();
      m_undoList.pop_back();
    }
  }
  catch (std::bad_alloc const& ex) {
    std::cerr << "Memory allocation: " << ex.what() << std::endl;
  }
  catch (std::exception const& ex) {
    std::cerr << "Exception: " << ex.what() << std::endl;
  }
  catch (...) {
    std::cerr << "Unhandled exception in Undo()!" << std::endl;
  }
}

void Control::Reset() {
  m_commands.clear();
  Undo(m_undoList.size());
}
```

5.3 Hexapod

5.3.1 Hexapod.h

```cpp
/* ----------------------------------------------------------------------
| Workfile : Hexapod.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */


#ifndef HEXAPOD_H
#define HEXAPOD_H


#include "Robot.h"
class Hexapod : public Robot {
public:
  //cTor
  Hexapod(std::string const& name) : Robot(name) {}

  //Prints all Information if valid ost is given
  virtual void Info(std::ostream& ost) override;
};

#endif//!HEXAPOD_H
```

5.3.2 Hexapod.cpp

```cpp
#include "Hexapod.h"

void Hexapod::Info(std::ostream& ost) {
  if (ost.good()) {
    ost << "Hexapod: ";
    Robot::Info(ost);
  }
}
```

5.4 Wheelbot

5.4.1 Wheelbot.h

```
/* -----------------------------------------------------------------------
| Workfile : Wheelpod.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ----------------------------------------------------------------------- */


#ifndef WHEELBOT_H
#define WHEELBOT_H

#include "Robot.h"

#include <ostream>

class Wheelbot : public Robot {
public:
  //cTor
  Wheelbot(std::string const& name) : Robot(name) {}

  //Prints all Information if valid ost is given
  virtual void Info(std::ostream& ost) override;
};

#endif//!WHEELBOT_H
```

5.4.2 Wheelbot.cpp

```
#include "Wheelbot.h"

void Wheelbot::Info(std::ostream& ost) {
  if (ost.good()) {
    ost << "Wheelbot: ";
    Robot::Info(ost);
  }
}
```

5.5 ICommand

5.5.1 ICommand.h

```
/* ----------------------------------------------------------------------
| Workfile : TurnLeft.h
| Description : [ HEADER ]
| Name : Daniel Weyrer    PKZ : S1820306044
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef ICOMMAND_H
#define ICOMMAND_H

#include "IRobot.h"
#include "Robot.h"

#include<memory>


class ICommand {
public:
  ICommand(std::shared_ptr<Robot> robot);
  virtual void Execute() = 0;
  virtual void Unexecute() = 0;

  using SPtr = std::shared_ptr<ICommand>;

protected:
  std::shared_ptr<Robot> m_robot;
};

#endif // !ICOMMAND_H
```

5.5.2 ICommand.cpp

```
#include "ICommand.h"

ICommand::ICommand(std::shared_ptr<Robot> robot) {
  if (robot == nullptr) {
    throw std::exception("Nullpointer in Constructor!");
  }
  m_robot = robot;
}
```

5.6 IRobot

5.6.1 IRobot.h

```cpp
/* -----------------------------------------------------------------------
| Workfile : IRobot.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ----------------------------------------------------------------------- */

#ifndef IROBOT_H
#define IROBOT_H

#include <iostream>

class IRobot {
public:
  // Prints current Robotinformation
  virtual void Info(std::ostream& ost) = 0;

  //dTor needed - Memory Leaks otherwise!
  virtual ~IRobot();

  //SharedPointer to IRobot for using Info()
  using SPter = std::shared_ptr<IRobot>;
};

#endif // !IROBOT_H
```

5.6.2 IRobot.cpp

```cpp
#include "IRobot.h"

IRobot::~IRobot() {}
```

5.7 Robot

5.7.1 Robot.h

```
/* ---------------------------------------------------------------------
| Workfile : Robot.h
| Description : [ HEADER ]
| Name : Viktoria Streibl   PKZ : S1810306013
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| --------------------------------------------------------------------- */

#ifndef ROBOT_H
#define ROBOT_H
#include <ostream>

enum class DIR {
  NORTH,
  SOUTH,
  EAST,
  WEST
};
std::ostream& operator<<(std::ostream& ost, DIR const dir);



#include "IRobot.h"
class Robot : public IRobot {
public:
  Robot() : m_posX{ 0 }, m_posY{ 0 }, m_direction{ DIR::NORTH }, m_name{ "" } {}
  Robot(std::string const& name) : m_posX{ 0 }, m_posY{ 0 }, m_direction{ DIR::NORTH }, m_name{ name
      } {}

  virtual void Info(std::ostream& ost) override;

  void SetDirection(DIR direction);
  DIR GetDirection();

  void SetPosX(int const pos);
  int GetPosX();

  void SetPosY(int const pos);
  int GetPosY();

  void SetName(std::string const& name);
  std::string GetName();

  using SPter = std::shared_ptr<Robot>;

private:
  std::string m_name;

  int m_posX;
  int m_posY;
  DIR m_direction;
};

#endif//!ROBOT_H
```

5.7.2 Robot.cpp

```
// Robot.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>
#include "Robot.h"



void Robot::Info(std::ostream& ost) {
  if (ost.good()) {
```

```cpp
    ost << m_name << ", " << "Pos(" << m_posX << "," << m_posY << "), " << "Direction(" <<
        m_direction << ")" << std::endl;
  }
}

void Robot::SetDirection(DIR direction) {
  m_direction = direction;
}

DIR Robot::GetDirection() {
  return m_direction;
}

void Robot::SetPosX(int const pos) {
  m_posX = pos;
}

int Robot::GetPosX() {
  return m_posX;
}

void Robot::SetPosY(int const pos) {
  m_posY = pos;
}

int Robot::GetPosY() {
  return m_posY;
}

void Robot::SetName(std::string const& name) {
  m_name = name;
}

std::string Robot::GetName() {
  return m_name;
}

std::ostream& operator<<(std::ostream& ost, DIR const dir) {
  if (ost.good()) {
    switch (dir) {
    case DIR::NORTH:
      ost << "NORTH";
      break;
    case DIR::SOUTH:
      ost << "SOUTH";
      break;
    case DIR::EAST:
      ost << "EAST";
      break;
    case DIR::WEST:
      ost << "WEST";
      break;
    default:
      break;
    }
  }
  return ost;
}
```

5.8 Forward

5.8.1 Forward.h

```
/* ----------------------------------------------------------------------
| Workfile : Forward.h
| Description : [ HEADER ]
| Name : Daniel Weyrer    PKZ : S1820306044
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef FORWARD_H
#define FORWARD_H


#include "ICommand.h"
#include "Robot.h"

class Forward : public ICommand {
public:
  Forward(std::shared_ptr<Robot> robot, int const distance) : ICommand{ robot }, m_distance{
      distance } {}
  virtual void Execute() override;
  virtual void Unexecute() override;

private:
  int m_distance;
};

#endif // !FORWARD_H
```

5.8.2 Forward.cpp

```
#include "Forward.h"

void Forward::Execute() {
  int posX = m_robot->GetPosX();
  int posY = m_robot->GetPosY();
  switch (m_robot->GetDirection()) {
  case DIR::NORTH:
    posY += m_distance;
    m_robot->SetPosY(posY);
    break;
  case DIR::SOUTH:
    posY -= m_distance;
    m_robot->SetPosY(posY);
    break;
  case DIR::EAST:
    posX += m_distance;
    m_robot->SetPosX(posX);
    break;
  case DIR::WEST:
    posX -= m_distance;
    m_robot->SetPosX(posX);
    break;
  default:
    break;
  }
}

void Forward::Unexecute() {
  m_distance *= -1;
  Execute();
  m_distance *= -1;
}
```

5.9 TurnLeft

5.9.1 TurnLeft.h

```
/* ----------------------------------------------------------------------
| Workfile : TurnLeft.h
| Description : [ HEADER ]
| Name : Daniel Weyrer    PKZ : S1820306044
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */


#ifndef TURNLEFT_H
#define TURNLEFT_H

#include "ICommand.h"
#include "Robot.h"

class TurnLeft : public ICommand {
public:
  TurnLeft(std::shared_ptr<Robot> robot) : ICommand{ robot } {}

  virtual void Execute() override;
  virtual void Unexecute() override;
};

#endif // !TURNLEFT_H
```

5.9.2 TurnLeft.cpp

```
#include "TurnLeft.h"

void TurnLeft::Execute() {
  DIR tmp = m_robot->GetDirection();
  switch (tmp) {
  case DIR::NORTH:
    tmp = DIR::WEST;
    break;
  case DIR::SOUTH:
    tmp = DIR::EAST;
    break;
  case DIR::EAST:
    tmp = DIR::NORTH;
    break;
  case DIR::WEST:
    tmp = DIR::SOUTH;
    break;
  default:
    break;
  }
  m_robot->SetDirection(tmp);
}

void TurnLeft::Unexecute() {
  DIR tmp = m_robot->GetDirection();
  switch (tmp) {
  case DIR::NORTH:
    tmp = DIR::EAST;
    break;
  case DIR::SOUTH:
    tmp = DIR::WEST;
    break;
  case DIR::EAST:
    tmp = DIR::SOUTH;
    break;
  case DIR::WEST:
    tmp = DIR::NORTH;
    break;
  default:
    break;
  }
  m_robot->SetDirection(tmp);
}
```

5.10 TurnRight

5.10.1 TurnRight.h

```
/* ----------------------------------------------------------------------
| Workfile : TurnRight.h
| Description : [ HEADER ]
| Name : Daniel Weyrer    PKZ : S1820306044
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef TURNRIGHT_H
#define TURNRIGHT_H

#include "ICommand.h"
#include "Robot.h"

class TurnRight : public ICommand {
public:
  TurnRight(std::shared_ptr<Robot> robot) : ICommand{ robot } {}
  virtual void Execute() override;
  virtual void Unexecute() override;
};

#endif // !TURNRIGHT_H
```

5.10.2 TurnRight.cpp

```
#include "TurnRight.h"

void TurnRight::Execute() {
  DIR tmp = m_robot->GetDirection();
  switch (tmp) {
  case DIR::NORTH:
    tmp = DIR::EAST;
    break;
  case DIR::SOUTH:
    tmp = DIR::WEST;
    break;
  case DIR::EAST:
    tmp = DIR::SOUTH;
    break;
  case DIR::WEST:
    tmp = DIR::NORTH;
    break;
  default:
    break;
  }
  m_robot->SetDirection(tmp);
}

void TurnRight::Unexecute() {
  DIR tmp = m_robot->GetDirection();
  switch (tmp) {
  case DIR::NORTH:
    tmp = DIR::WEST;
    break;
  case DIR::SOUTH:
    tmp = DIR::EAST;
    break;
  case DIR::EAST:
    tmp = DIR::NORTH;
    break;
  case DIR::WEST:
    tmp = DIR::SOUTH;
    break;
  default:
    break;
  }
  m_robot->SetDirection(tmp);
}
```

5.11 MacroMovement

5.11.1 MacroMovement.h

```cpp
/* ----------------------------------------------------------------------
| Workfile : Forward.h
| Description : [ HEADER ]
| Name : Daniel Weyrer    PKZ : S1820306044
| Date : 13.01.2020
| Remarks : -
| Revision : 0
| ---------------------------------------------------------------------- */

#ifndef MACROMOVEMENT_H
#define MACROMOVEMENT_H

#include "Control.h"
#include "ICommand.h"
#include "Robot.h"

class MacroMovement : public ICommand {
public:
  MacroMovement(std::shared_ptr<Robot> robot, Control::TcmdList const& movement) : ICommand{ robot
      }, m_movement{ movement }{}
  virtual void Execute() override;
  virtual void Unexecute() override;

private:
  Control::TcmdList m_movement;
  Control::TcmdList m_undoMovement;
};

#endif // !MACROMOVEMENT_H
```

5.11.2 MacroMovement.cpp

```cpp
#include "MacroMovement.h"

void MacroMovement::Execute() {
  for (auto elem : m_movement) {
    if (elem == nullptr) {
      throw std::exception("Nullptr in given commandlist of MacroMovement!");
    }
    elem->Execute();
    m_undoMovement.emplace_back(elem);
  }
}

void MacroMovement::Unexecute() {
  size_t i = 0;
  while (i < m_undoMovement.size() && m_undoMovement.size() > 0) {
    m_undoMovement.back()->Unexecute();
    m_undoMovement.pop_back();
  }
}
```

5.12 Object

5.12.1 Object.h

```
/* -----------------------------------------------------------------------
| Workfile : Object .h
| Description : [ HEADER ] Class for the global Object
| Name : Viktoria Streibl     PKZ : S1810306013
| Date : 04.11.2019
| Remarks : -
| Revision : 0
| ----------------------------------------------------------------------- */
#ifndef OBJECT_INCLUDED
#define OBJECT_INCLUDED
class Object {
public:
  // ----------------------------------------------------------------------
  // Destructor of the class Object
  // -----------------------------------------------------------------------
  virtual ~Object() = default;
protected:
  // ----------------------------------------------------------------------
  // Constructor of the class Object
  // No object of the class can be created directly
  // -----------------------------------------------------------------------
  Object() = default;
};
#endif //OBJECT_INCLUDED
```

5.13 TestDriver

5.13.1 TestDriver.cpp

```cpp

#include "Client.h"

using namespace std;
int main() {
  Client c;
  c.Test();
}
```

5.14 output

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Detector\vld.ini
Visual Leak Detector Version 2.5.1 installed.
Exception in AddCommand: Given Command is nullptr!
Exception during Execution: Nullptr in given commandlist of MacroMovement!
Hexapod: Robert, Pos(400,0), Direction(EAST)
Wheelbot: Harry, Pos(300,-200), Direction(SOUTH)
Hexapod: Robert, Pos(0,0), Direction(NORTH)
Wheelbot: Harry, Pos(600,-200), Direction(WEST)
No memory leaks detected.
Visual Leak Detector is now exiting.
```