LOICH KAMDOUM DEAMENI =>Mat-Nr: 506520
NDAME EKOBE HUGUE BENJAMIN => Mat-Nr: 495921

Uebungsblatt 6:

Ubung 1

1) Das Programm ruft die zwei Klasse untereinander an. Zuerst wird die Klasse P angerufen, dann wird diese Klasse (Wenn es keinen Fehler gibt) die Klasse Ball anrufen. Dieses Programm benehmt sich wie ein rekursives Programm, also ruft sich selber ohne Ende an.

Uebung_1.cpp

```
#include <iostream>
#include <exception>

//Deklaration der Klasse Ball mit der Erbschaft exception
class Ball: public std::exception
{

};

//Deklaration der Klasse P
class P
{
	public:
	P *target;
	std::string msg;

	// Konstrutor
	P(P *target_, std::string msg_){
		target = target_;
		msg = msg_;
	}

	// Destruktor
	~P(){
		target = nullptr;
		msg = nullptr;
	}

	// Funktion aim
	void aim(Ball *ball){
		// Exception
		try{
			throw Ball();
		}
		catch(Ball &b){
			std::cout << msg << std::endl;
			this->target->aim(&b);
		}
	}
```

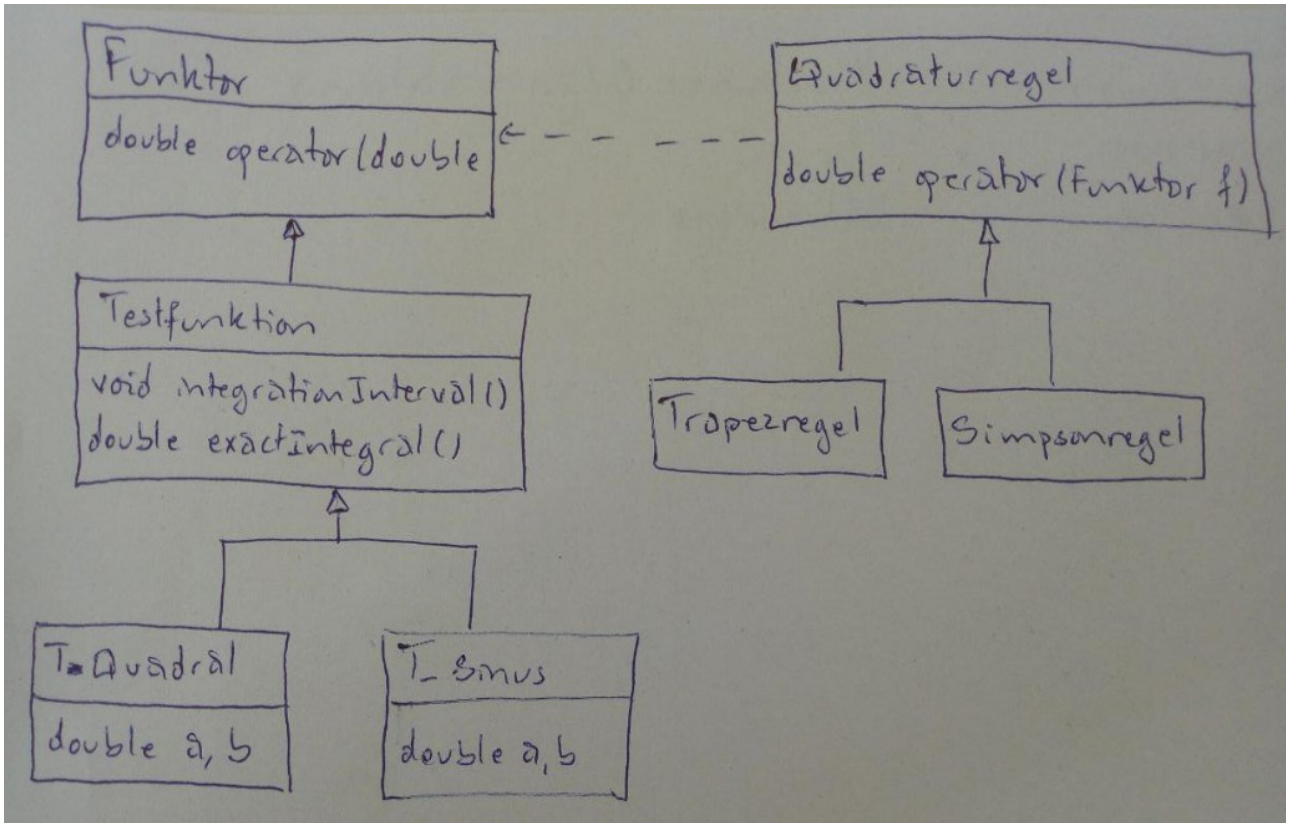
```
};

int main();

int main()
{
	P *parent = new P(nullptr, "Father");
	P *child = new P(parent, "Son");

	parent->target = child;
	parent->aim(new Ball());
}
```

Uebung 2 und 3



funktor.h

```
#ifndef FUNKTOR_H
#define FUNKTOR_H

class Funktor
{
public:
   virtual double operator () (double f) = 0;
};

#endif // FUNKTOR_H
```

integration.h

```
#ifndef INTEGRATION_H
#define INTEGRATION_H

#include <math.h>

#include "quadraturregel.h"
#include "funktor.h"
#include "testfunktion.h"

class Integration
{
private:

public:
   double konvergenceOrdnung(TestFunktion &, Quadraturregel &, Quadraturregel &);
   void printTestQuadrature(double, double, double, TestFunktion &, Quadraturregel &,
Quadraturregel &);
   void testTrapezregel(double, double, double, TestFunktion &);
   void testSimpsonregel(double, double, double, TestFunktion &);
};

#endif // INTEGRATION_H
```

integration.cpp

```
#include <iostream>

#include "integration.h"
#include "trapezregel.h"
#include "simpsonregel.h"

double Integration::konvergenceOrdnung(TestFunktion &f, Quadraturregel &q1, Quadraturregel
&q2)
{
   double EOC = 0, EN = 0, E2N = 0;

   EN = f.exactIntegral() - q1(f);
   E2N = f.exactIntegral() - q2(f);

   if (EN != 0 && E2N != 0)
   {
      EOC = std::log(EN/E2N)/std::log(2);
   }

   return EOC;
}
```

```
void Integration::printTestQuadrature(double a_, double b_, double size_, TestFunktion &f,
Quadraturregel &q1, Quadraturregel &q2)
{
   std::cout << "[a,b] = [" << a_ << "," << b_ << "]" << " size = " << size_ << std::endl;
   std::cout << "" << std::endl;
   std::cout << "Numerische Integral " << q1(f) << std::endl;
   std::cout << "Exactes Integral " << f.exactIntegral() << std::endl;
   std::cout << "Konvergence Zuordnung " << Integration::konvergenceOrdnung(f, q1, q2) <<
std::endl;
   ;
   std::cout << std::endl;
}

void Integration::testTrapezregel(double a_, double b_, double size_, TestFunktion &f)
{
   Trapezregel q(a_, b_, size_);
   Trapezregel q2n(a_, b_, 2 * size_);
   printTestQuadrature(a_, b_, size_, f, q, q2n);
}

void Integration::testSimpsonregel(double a_, double b_, double size_, TestFunktion &f)
{
   Simpsonregel q(a_, b_, size_);
   Simpsonregel q2n(a_, b_, 2 * size_);
   printTestQuadrature(a_, b_, size_, f, q, q2n);
}
```

quadraturregel.h

```
#ifndef QUADRATURREGEL_H
#define QUADRATURREGEL_H

#include "funktor.h"

class Quadraturregel
{
public:
   virtual double operator () (Funktor &) = 0;
};

#endif // QUADRATURREGEL_H
```

simpsonregel.h

```
#ifndef SIMPSONREGEL_H
#define SIMPSONREGEL_H

#include <math.h>

#include "quadraturregel.h"
```

```
class Simpsonregel: public Quadraturregel
{
private:
    double qSimpson(Funktor &, double, double);

public:
    //Variable
    double a, b;
    int size;

    //Funktionen
    Simpsonregel(double, double, int);
    ~Simpsonregel();
    double operator() (Funktor &) override;
};

#endif // SIMPSONREGEL_H
```

simpsonregel.cpp

```
#include "simpsonregel.h"

Simpsonregel::Simpsonregel(double a_, double b_, int size_)
{
    a = a_;
    b = b_;
    size = size_;
}

Simpsonregel::~Simpsonregel()
{
    a = 0;
    b = 0;
    size = 0;
}

double Simpsonregel::qSimpson(Funktor &f, double a_, double b_)
{
    return (b_ - a_) * (f(a_) + (4 * f((a_ + b_) / 2) + f(b_))) / 6;
}

double Simpsonregel::operator()(Funktor &f) //override
{
    double h = (b - a) / size;

    double s = a;
    double result = 0;

    for (size_t i = 0; i < size; i++)
    {
```

```
        result += Simpsonregel::qSimpson(f, s, s + h);
        s = s + h;
    }

    return result;
}
```

t_quadrat.h

```
#ifndef T_QUADRAT_H
#define T_QUADRAT_H

#include <math.h>

#include "testfunktion.h"
#include "funktor.h"

class T_Quadrat: public TestFunktion
{
private:
    double a, b;

public:
    //Variable

    //Funktionen
    T_Quadrat() = default;
    T_Quadrat(double, double);

    double operator () (double) override;

    void integrationIntegration(double &, double &) const;
    double exactIntegral() const override;
};

#endif // T_QUADRAT_H
```

t_quadrat.cpp

```
#include <math.h>

#include "t_quadrat.h"

T_Quadrat::T_Quadrat(double a_, double b_)
{
    a = a_;
    b = b_;
}

double T_Quadrat::operator()(double t)
{
```

```
    return (2 * pow(t, 2)) + 5;
}

double T_Quadrat::exactIntegral() const
{
    return (((2 * pow(b, 3)) / (3 + 5 * b)) - ((2 * pow(a, 3)) / (3 + 5 * a)));
}
```

t_sinus.h

```
#ifndef T_SINUS_H
#define T_SINUS_H

#include <math.h>

#include "testfunktion.h"
#include "funktor.h"

class T_Sinus : public TestFunktion
{
private:
    double a, b;

public:
    //Variable

    //Funktionen
    T_Sinus() = default;
    T_Sinus(double, double);

    double operator()(double) override;

    void integrationIntegration(double &, double &) const;
    double exactIntegral() const override;
};

#endif // T_SINUS_H
```

t_sinus.cpp

```
#include "t_sinus.h"
#include <math.h>

T_Sinus::T_Sinus(double a_, double b_)
{
    a = a_;
    b = b_;
}

double T_Sinus::operator()(double t_)
```

```
{
   return ((t_ * std::sin(t_)) / M_PI);
}

double T_Sinus::exactIntegral() const
{
   return ((std::sin(b) - b * std::cos(b)) - (std::sin(b) - b * std::cos(b))) / M_PI;
}
```

testfunktion.h

```
#ifndef TESTFUNKTIONCLASS_H
#define TESTFUNKTIONCLASS_H

#include <math.h>

#include "quadraturregel.h"
#include "funktor.h"

class TestFunktion: public Funktor
{
public:
   //Variable

   //Funktionen
   TestFunktion() = default;
   ~TestFunktion() = default;

   void integrationIntervall(double &, double &) const;
   virtual double exactIntegral() const = 0;
};

#endif // TESTFUNKTIONCLASS_H
```

testfunktion.cpp

```
#include <iostream>

#include "testfunktion.h"

void TestFunktion::integrationIntervall(double &l_, double &r_) const
{
   std::cout << "[l, r] = " << l_ << " " << r_ << std::endl;
}

double TestFunktion::exactIntegral() const
{
   return 0;
}
```

trapezregel.h

```
#ifndef TRAPEZREGEL_H
#define TRAPEZREGEL_H

#include <math.h>

#include "quadraturregel.h"

class Trapezregel : public Quadraturregel
{
private:
   double qTrapez(Funktor &, double, double);

public:
   //Variable
   double a, b;
   int size;

   //Funktionen
   Trapezregel(double, double, int);
   ~Trapezregel();
   double operator() (Funktor &) override;
};

#endif // TRAPEZREGEL_H
```

trapezregel.cpp

```
#include "trapezregel.h"

Trapezregel::Trapezregel(double a_, double b_, int size_)
{
   a = a_;
   b = b_;
   size = size_;
}

Trapezregel::~Trapezregel()
{
   a = 0;
   b = 0;
   size = 0;
}

double Trapezregel::qTrapez(Funktor &f, double a_, double b_)
{
   return ((b_ - a_) * (f(a_) + f(b_))) / 2;
}
```

```
double Trapezregel::operator() (Funktor &f) //override
{
   double h = (b - a) / size;

   double s = a;
   double result = 0;

   for (size_t i = 0; i < size; i++)
   {
      result += Trapezregel::qTrapez(f, s, s + h);
      s = s + h;
   }

   return result;
}
```

main.cpp

```
#include <iostream>

#include "funktor.h"
#include "integration.h"
#include "quadraturregel.h"
#include "simpsonregel.h"
#include "t_quadrat.h"
#include "t_sinus.h"
#include "testfunktion.h"
#include "trapezregel.h"
#include "testfunktion.h"

int main();

int main()
{
   int size = 100;
   double a = 0.;
   double b = 0.;

   a = -3.;
   b = 13.;

   T_Quadrat t_quadrat(a, b);
   Integration *integrall;

   std::cout << "Trapezregel 2*t*t+5" << std::endl;
   integrall->testTrapezregel(a, b, size, t_quadrat);
   std::cout << "simpson 2*t*t+5" << std::endl;
   integrall->testSimpsonregel(a, b, size, t_quadrat);

   a = 0.;
   b = 2 * M_PI;
```

```
    T_Sinus t_sinus(a, b);
    std::cout << "Trapezregel t*sin(t)/M_PI" << std::endl;
    integrall->testTrapezregel(a, b, size, t_sinus);
    std::cout << "Simpson t*sin(t)/M_PI" << std::endl;
    integrall->testSimpsonregel(a, b, size, t_sinus);

    return 0;
}
```