Software Engeneering Analysis Sientific Report

Kendra Birringer (1229372)
Nader Cacace (1208115)
Steffen Hanzlik (1207417)
Marco Peluso (1228849)
Svetozar Stojanovic (1262287)

Frankfurt University of Applied Sciences

November 2, 2019

Contents

List of Figures

Figure 1: Fahrzeug Program

1 Exercise 2

1.1 Fahrzeug

```cpp
void Fahrzeug::setName(const char *n)
{
     if (name != nullptr) {
      delete name;
      name = nullptr; // handling dangling pointer to freed memory
     }

     if (n != nullptr) { // handling empty pointer as parameter, preventing
         strlen from crashing
      name = new char[strlen(n)+1];
      strcpy(name, n);
     }
}



Fahrzeug& Fahrzeug::operator=(const Fahrzeug &other) {
   delete[] name;

   if (other.name != nullptr) { // handling empty pointer as parameter,
        preventing strlen from crashing
      name = new char[strlen(other.name) + 1];
      strcpy(name, other.name);
   }
   return *this;
}
```

Figure 2: Header of EBook Program

2 Exercise 3

2.1 EBook Headerfile

```cpp
#ifndef _EBOOK_H_
#define _EBOOK_H_

#include <string>
#include <iostream>

using namespace std;

class EBook {
private:
   string title, content;
public:
   EBook();
   EBook(string title, string content);
   void setTitle(string title);
   string getTitle() const;
   void setContent(string content);
   string getContent() const;
   void print() const;
   friend ostream &operator<<(ostream &output, const EBook &book);
};

#endif
```

In the header file the declaration of the private and the public members of the EBook class takes place.
The private members are two Strings named 'title' and 'content'. The public members of the class EBook are the default constructor and an overloaded constructor with the parameters title and content.
Then we also declared the public 'getter' and 'setter' methods. Furthermore we need a print method and we must overload the &operator<< method.

Figure 3: Implementation of EBook class

2.2 Implementation of the EBook class

```cpp
#include "eBook.h"
#include <iostream>

using namespace std;

EBook::EBook() : title(""), content("") {};

EBook::EBook(string title, string content) : title(title), content(content) {};

void EBook::setTitle(string title) {
   if (title != "") {
      this->title = title;
   } else {
      cout << "Title not set!" << endl;
   }
}
string EBook::getTitle() const {
   return this->title;
}
void EBook::setContent(string content) {
   if (content != "") {
      this->content = content;
   } else {
      cout << "Content not set!" << endl;
   }
}
string EBook::getContent() const {
   return this->content;
}

void EBook::print() const {
   cout << "Title: " << this->title << '\n';
   cout << "Content: " << this->content << '\n';
}

ostream & operator<<(ostream &output, const EBook &book) {
   book.print();
   return output;
}
```

In the Ebook.cpp file we implemented the declared methods of Ebook.h. First we implemented the standard constructor initializing the member variables 'title' and 'content' using an initializer list. Furthermore we implemented an overloaded constructor of Bbook with the arguments 'title' and 'content' and initialized the 'title' and 'content' with the passed arguments 'title' and 'content'. Then we implemented the 'getter' and 'setter' methods. Also we implemented the 'print' method. This method is for printing the 'title' and the 'content' and we overloaded the &operater<< method.

Figure 4: Main class of Ebook program

Figure 5: Header of Box class

2.3 Main class

```cpp
#include <iostream>
#include "eBook.h"

int main() {
   EBook book("Brown Fox", "The quick brown fox jumps over the lazy
       dog.");
   std::cout << book;

   return 0;
}
```

The main method executes the EBook class.

3 Exercise 4

3.1 Box Headerfile

```cpp
#ifndef _BOX_H_
#define _BOX_H_

class Box {
private:
   double xMin, xMax, yMin, yMax;
public:

   Box();
   double getXMin() const;
   double getXMax() const;
   double getYMin() const;
   double getYMax() const;
   void setXMax(double val);
   void setXMin(double val);
   void setYMin(double val);
   void setYMax(double val);
   friend Box operator+(Box left, Box right);
   void print() const;
};
#endif
```

In the header file the declaration of the private and the public members of the Box class takes place.

The private members are four double type variable named 'xMin', 'xMax', 'yMin' and 'yMax'.
One public member of the class Box is the default constructor.
Then we also declared the public 'getter' and 'setter' methods. Furthermore also we need a 'print' method and we must overload the $+operator <<$ method.

Figure 6: Header of Circle class

Figure 7: Header of Form class

3.2 Circle Headerfile

```
#ifndef _CIRCLE_H_
#define _CIRCLE_H_

#include "Form.h"

class Circle : public Form
{
private:
   double radius;
public:
   Circle();
   Circle(double rad);
   void move(double dX, double dY);
   void setUpBox();
private:
   void moveBox(double dX = 0, double dY = 0);
};
#endif
```

In the header file of the Circle class we declare the private and public members. One of the private members is a double type variable named 'radius'. The other private member is a method named 'moveBox' with the Arguments 'dX' and 'dY'. The public members are the standard and a overloaded constructor with the parameter 'rad' which is a double type variable. The other public members are the 'move' method with the two arguments 'dX' and 'dY' and a 'setUp' method.

3.3 Form Headerfile

```
#ifndef _FORM_H_
#define _FORM_H_

#include "Box.h"

class Form {
private:
   double xCenter, yCenter;
protected:
   Box box;
public:
   Form();
   void move(double dX, double dY);
   Box &getBoxRef();
};
#endif
```

Figure 8: Header of Rectangle class

Figure 9: Box class Implementation

In the header file the declaration of the private and public members of the 'Form' class takes place. The private members a two double type variables named 'xCenter' and 'yCenter' The protected member is an object of the Box class named 'box'. The public members of the class 'Form' are the default constructor, a method named 'move' with the double type arguments 'dX' and 'dY' and a 'getBoxRef' method.

3.4 Rectangle Headerfile

```
#ifndef _RECTANGLE_H_
#define _RECTANGLE_H_

#include "Form.h"

class Rectangle: public Form {
private:
   double width, height;

public:
   Rectangle();
   Rectangle(double h, double w);
   //MOVE FOR RECT
   void move(double dX, double dY);
   void setUpBox();
private:
   void moveBox(double dX = 0, double dY = 0);
};
#endif
```

In the header file the declaration of the private and the public members of the Form class takes place.
The private members are two double with the name 'xCenter' and 'yCenter'. The protected member is a Object Box with the name 'box'. The public members of the class Form are the default constructor.
Also we need a move method with the arguments 'dX' and 'dY' and a method getBoxRef that returns a reference to the attribute Box.

Figure 10: Box class Implementation

3.5 Implementation of the Box class

```cpp
#include "Box.h"
#include <iostream>
#include <algorithm>

using namespace std;

Box::Box() : xMin(0.0), xMax(0.0), yMin(0.0), yMax(0.0) {

}

double Box::getXMin() const {
   return xMin;
}

void Box::setXMax(double val) {
   this->xMax = val;
}

double Box::getXMax() const {
   return xMax;
}

void Box::setXMin(double val) {
   this->xMin = val;
}

double Box::getYMin() const {
   return yMin;
}

void Box::setYMin(double val) {
   this->yMin = val;
}

double Box::getYMax() const {
   return yMax;
}

void Box::setYMax(double val) {
   this->yMax = val;
}

void Box::print() const {
   cout << "xMax: " << xMax << endl;
   cout << "xMin: " << xMin << endl;
   cout << "yMax: " << yMax << endl;
   cout << "yMin: " << yMin << endl;
}

Box operator+(Box left, Box right) {
   Box newLeft, newRight;
   if (left.getXMax() > right.getXMax()) {
      newLeft = right;
      newRight = left;
   } else {
      newLeft = left;
      newRight = right;
   }

   Box result;

   //check if the boxes collide
   if (right.getXMin() < left.getXMax() && right.getYMin() < left.getYMax()) {
      result.setXMin(min(newLeft.getXMin(), newRight.getXMin()));
      result.setXMax(max(newLeft.getXMax(), newRight.getXMax()));
      result.setYMin(min(newLeft.getYMin(), newRight.getYMin()));
      result.setYMax(max(newLeft.getYMax(), newRight.getYMax()));
      return result;
   } else {
      cout << "The boxes of these two objects don't collide." << '\n';
   }

}
```

In the Box class we implemented getter and setter Methods. Also we implemented the print method to show the Min and Max coordinate. Then we overload the operator+ method.

Figure 11: Circle class Implementation

3.6 Circle Class

```
#include "Circle.h"

Circle::Circle() : radius(0.0) {
  Form();
  this->box.setXMax(0.0);
  this->box.setXMin(0.0);
  this->box.setYMax(0.0);
  this->box.setYMin(0.0);
}

Circle::Circle(double rad) : radius(rad) {

}

void Circle::setUpBox() {
  this->box.setXMax(this->radius);
  this->box.setXMin(-this->radius);
  this->box.setYMax(this->radius);
  this->box.setYMin(-this->radius);
}

void Circle::move(double dX, double dY) {
  Form::move(dX, dY);
  moveBox(dX, dY);
}

void Circle::moveBox(double dX, double dY) {
  this->box.setXMax(box.getXMax() + dX);
  this->box.setXMin(box.getXMin() + dX);
  this->box.setYMax(box.getYMax() + dY);
  this->box.setYMin(box.getYMin() + dY);
}
```

In the Circle class we implemented the default constructor and the overloaded constructor aswell. Furthermore we set all coordinates for the Box that surround the circle object in the setUpBox method. We also wrote a move method to move the circle and a moveBox method that gets called from the move method to shift the Box to the same place.

Figure 12: Rectangle class Implementation

3.7 Implementation of the Rectangle Class

```
#include "Rectangle.h"

Rectangle::Rectangle() : width(0.0), height(0.0) {
   Form();
   this->box.setXMax(0.0);
   this->box.setXMin(0.0);
   this->box.setYMax(0.0);
   this->box.setYMin(0.0);
}

Rectangle::Rectangle(double h, double w) : width(w), height(h) {

}

void Rectangle::move(double dX, double dY)
{
   Form::move(dX, dY);
   moveBox(dX, dY);
}

void Rectangle::moveBox(double dX, double dY) {
   this->box.setXMax(box.getXMax()+dX);
   this->box.setXMin(box.getXMin()+dX);
   this->box.setYMax(box.getYMax()+dY);
   this->box.setYMin(box.getYMin()+dY);
}

void Rectangle::setUpBox() {
   this->box.setXMax(width / 2);
   this->box.setXMin(-width / 2);
   this->box.setYMax(height / 2);
   this->box.setYMin(-height / 2);
}
```

In the Rectangle class we implemented the default constructor and the overloaded constructor. Besides we initialized the box in the setUpBox method. The move method calls the move method from the Form class and on the other hand the moveBox method to move the hole object to a other position.

Figure 13: Main class Implementation

3.8 Implementation of the Main Class

```
#include <iostream>
#include <string>
#include "Circle.h"
#include "Rectangle.h"

using namespace std;

//checks if the user typed 'circle', returns bool
bool inputIsCircle(string);

//checks if the user typed 'rectangle', returns bool
bool inputIsRect(string);

//asks for needed values and calls circle constructor
Circle* circleCreator(bool isTrue);

//asks for needed values and calls rectangle constructor
Rectangle* rectCreator(bool isTrue);
Box addBoxes(Circle* c1, Circle* c2, Rectangle* r1, Rectangle* r2);

int main() {

   //arguments for move(...) function
   double movX, movY;

   string prompt = "";

   std::cout << "________________________________________" << endl;

   Circle *circle1 = NULL;
   Rectangle *rect1 = NULL;

   cout << "Enter first form (rectangle or circle): ";
   cin >> prompt;

   if (inputIsCircle(prompt)) {
      circle1 = circleCreator(inputIsCircle(prompt));
      circle1->getBoxRef().print();

      cout << "Move Circle in X direction for: ";
      cin >> movX;
      cout << "Move Circle in Y direction for: ";
      cin >> movY;

      circle1->move(movX, movY);
      cout << "After Move is called: " << endl;
      circle1->getBoxRef().print();
   } else if (inputIsRect(prompt)) {
      rect1 = rectCreator(inputIsRect(prompt));
      rect1->getBoxRef().print();

      cout << "Move Rectangle in X direction for: ";
      cin >> movX;
      cout << "Move Rectangle in Y direction for: ";
      cin >> movY;

      rect1->move(movX, movY);
      cout << "After Move is called: " << endl;
      rect1->getBoxRef().print();
   }

   Circle *circle2 = NULL;
   Rectangle *rect2 = NULL;
   cout << "Enter second form (rectangle or circle): ";
   cin >> prompt;
   if (inputIsCircle(prompt)) {
      circle2 = circleCreator(inputIsCircle(prompt));
      circle2->getBoxRef().print();

      cout << "Move Circle in X direction for: ";
      cin >> movX;
      cout << "Move Circle in Y direction for: ";
      cin >> movY;

      circle2->move(movX, movY);
      cout << "After Move is called: " << endl;
      circle2->getBoxRef().print();
   } else if (inputIsRect(prompt)) {
      rect2 = rectCreator(inputIsRect(prompt));
      rect2->getBoxRef().print();

      cout << "Move Rectangle in X direction for: ";
      cin >> movX;
      cout << "Move Rectangle in Y direction for: ";
      cin >> movY;

      rect2->move(movX, movY);
      cout << "After Move is called: " << endl;
      rect2->getBoxRef().print();
   }

   //ADD BOUNDING BOXES AND PRODUCE NEW ONE AS SUM
   Box boundingBox;

   cout << "Bounding Box: " << endl;
   boundingBox = addBoxes(circle1, circle2, rect1, rect2);
   if (!(boundingBox.getXMax() == 0.0 && boundingBox.getXMin() == 0.0 &&
       boundingBox.getYMin() == 0.0 && boundingBox.getYMax() == 0.0)) {
      boundingBox.print();
   }

   cout << "________________________________________" << endl;

   delete circle1, rect1, circle2, rect2;

   return 0;
}
```

```
Box addBoxes(Circle* c1, Circle* c2, Rectangle* r1, Rectangle* r2) {

   Box result;
   if (c1 == NULL && c2 == NULL) {
      result = r1->getBoxRef() + r2->getBoxRef();
      return result;
   } else if (c1 == NULL && r2 == NULL) {
      result = r1->getBoxRef()+ c2->getBoxRef();
      return result;
   } else if (r1 == NULL && c2 == NULL) {
      result = c1->getBoxRef()+ r2->getBoxRef();
      return result;
   } else if (r1 == NULL && r2 == NULL) {
      result = c1->getBoxRef() + c2->getBoxRef();
      return result;
   }

   return result;
}

bool inputIsCircle(string prompt) {
   string circle = "circle";
   bool result = false;
   if (prompt.compare(circle) == 0) {
      result = true;
   } else {
      return result;
   }
   return result;
}

bool inputIsRect(string prompt) {
   string rect = "rectangle";
   bool result = false;
   if (prompt.compare(rect) == 0) {
      result = true;
   } else {
      return result;
   }

   return result;
}
Circle* circleCreator(bool isTrue) {
   if (isTrue) {
      double rad;

      cout << "Enter radius: ";
      cin >> rad;
      Circle *circle = new Circle(rad);
      circle->setUpBox();
      return circle;
   } else {
      return NULL;
   }
}
Rectangle* rectCreator(bool isTrue) {
   if (isTrue) {
      double h, w;
      cout << "Enter height: ";
      cin >> h;
      cout << "Enter width: ";
      cin >> w;
      Rectangle *rect = new Rectangle(h, w);
      rect->setUpBox();
      return rect;
   } else {
      return NULL;
   }
}
```

In the main class implementation every class functions are build and work together, to build a box with the user input coordinates, draw a form like a circle or a rectangle, and after the build move it with the user input passed coordinates. After the move finish and close the box and the program.