Software Engineering Analysis Scientific 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

Listing 1: My Caption

```
1  void Fahrzeug::setName(const char *n)
2  {
3      if (name != nullptr) {
4       delete name;
5       name = nullptr; // handling dangling pointer to freed memory
6      }
7
8      if (n != nullptr) { // handling empty pointer as parameter, preventing
           strlen from crashing
9       name = new char[strlen(n)+1];
10      strcpy(name, n);
11     }
12 }
13
14
15
16 Fahrzeug& Fahrzeug::operator=(const Fahrzeug &other) {
17   delete[] name;
18
19   if (other.name != nullptr) { // handling empty pointer as parameter,
         preventing strlen from crashing
20     name = new char[strlen(other.name) + 1];
21     strcpy(name, other.name);
22   }
23   return *this;
24 }
```

Figure 2: Header of EBook Program

2 Exercise 3

2.1 EBook Headerfile

Listing 2: My Caption

```
#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

Listing 3: My Caption

```
#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

Figure 4: Main class of Ebook program

Figure 5: Header of Form class

we overloaded the &operater<< method.

2.3 Main class

Listing 4: My Caption

```
#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 Form Headerfile

Listing 5: My Caption

```
#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
```

This header file declares a class 'Form', which is the base class for classes 'Circle' and 'Rectangle'. The private members of this class are two double variables 'xCenter' and 'yCenter', which store coordinates for the center of the form. The protected member is a Box type object called 'box'. Because of this, every

class that derives from 'Form' class has its own 'box' member and it cannot be accessed directly outside of this class. As public members we have a default constructor called 'Form()', a 'move' method that changes the coordinates of the form depending on the passed parameters 'dX' and 'dY', and a method called 'getBoxRef()' that returns a reference to the 'Box' type object in this class.

Figure 6: Form class Implementation

Figure 7: Header of Box class

3.2 Implementation of Form class

Listing 6: My Caption

```
#include "Form.h"

Form::Form() : xCenter(0.0), yCenter(0.0) {

}

void Form::move(double dX, double dY) {
  this->xCenter += dX;
  this->yCenter += dY;
}

Box & Form::getBoxRef() {
  return box;
}
```

3.3 Box Headerfile

Listing 7: My Caption

```
#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 'Box' class is the representation of the bounding for each form. It is included as an object in 'Form', 'Circle' and 'Rectangle' classes. The private members of this class are variables 'xMin', 'xMax', 'yMin' and 'yMax'. The public members are typical getter und setter conventional methods for this class, also an overloaded constructor 'Box()' that zero-initializes the private members, a 'print' method to display the bounding box values for this class, and an overloaded '+' operator declared as a friend function. It is important to declare the overloaded operator as a friend function because we want to directly access private data members. Also, this overloaded operator's purpose is to create a new bounding box only if the two 'Box' type objects passed as arguments.

Figure 8: Box class Implementation

3.4 Implementation of the Box class

Listing 8: My Caption

```
#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;
}

```

```
50  Box operator+(Box left, Box right) {
51     Box newLeft, newRight;
52     if (left.getXMax() > right.getXMax()) {
53        newLeft = right;
54        newRight = left;
55     } else {
56        newLeft = left;
57        newRight = right;
58     }
59
60     Box result;
61
62     //check if the boxes collide
63     if (right.getXMin() < left.getXMax() && right.getYMin() < left.getYMax()) {
64        result.setXMin(min(newLeft.getXMin(), newRight.getXMin()));
65        result.setXMax(max(newLeft.getXMax(), newRight.getXMax()));
66        result.setYMin(min(newLeft.getYMin(), newRight.getYMin()));
67        result.setYMax(max(newLeft.getYMax(), newRight.getYMax()));
68        return result;
69     } else {
70        cout << "The boxes of these two objects don't collide." << '\n';
71     }
72
73  }
```

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.

Here are all methods from the header file implemented. The overloaded operator '+' checks if the bounding boxes collide and if so, displays a message to the user.

Figure 9: Header of Circle class

3.5 Circle Headerfile

Listing 9: My Caption

```
1  #ifndef _CIRCLE_H_
2  #define _CIRCLE_H_
3
4  #include "Form.h"
5
6  class Circle : public Form
7  {
8  private:
9     double radius;
10 public:
11    Circle();
12    Circle(double rad);
13    void move(double dX, double dY);
14    void initBox();
15 private:
16    void moveBox(double dX = 0, double dY = 0);
17 };
18 #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 'initBox' method.

Before declaring the 'Circle' class we must first include the base class 'Form' (see the line number 4). This header file declares a 'Circle' class, derived from the 'Form' class. In the private section we have a double variable 'radius' and a function 'moveBox'. Variable 'radius' stores the radius of the circle and the function 'moveBox' changes the position of the bounding box for this form. The public methods are two overloaded constructors 'Circle()' and 'Circle(double rad)', a 'move' method that changes the position of the circle, and a method 'initBox' that initializes the bounding box.

Figure 10: Circle class Implementation

3.6 Implementation of Circle class

Listing 10: My Caption

```
1  #include "Circle.h"
2
3  Circle::Circle() : radius(0.0) {
4     Form();
5     this->box.setXMax(0.0);
6     this->box.setXMin(0.0);
7     this->box.setYMax(0.0);
8     this->box.setYMin(0.0);
9  }
10
11 Circle::Circle(double rad) : radius(rad) {
12
13 }
14
15 void Circle::initBox() {
16    this->box.setXMax(this->radius);
17    this->box.setXMin(-this->radius);
18    this->box.setYMax(this->radius);
19    this->box.setYMin(-this->radius);
20 }
21
22 void Circle::move(double dX, double dY) {
23    Form::move(dX, dY);
24    moveBox(dX, dY);
25 }
26
27 void Circle::moveBox(double dX, double dY) {
28    this->box.setXMax(box.getXMax() + dX);
29    this->box.setXMin(box.getXMin() + dX);
30    this->box.setYMax(box.getYMax() + dY);
31    this->box.setYMin(box.getYMin() + dY);
32 }
```

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 initBox 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.

The first overloaded constructor 'Circle()' just initializes the 'radius' variable with zero. The second overloaded constructor 'Circle(double rad)' initializes the 'radius' variable with the value of the parameter 'rad'. Method 'initBox' of the Circle class initializes the values of the 'box' object of the 'Circle' class. The overloaded method 'move' calls the 'move' function of its base class 'Form' and the private method 'moveBox' that changes the values of the bounding box.

This makes the bbox follow the circle.

Figure 11: Header of Rectangle class

3.7 Rectangle Headerfile

Listing 11: My Caption

```
1  #ifndef _RECTANGLE_H_
2  #define _RECTANGLE_H_
3
4  #include "Form.h"
5
6  class Rectangle: public Form {
7  private:
8     double width, height;
9
10 public:
11    Rectangle();
12    Rectangle(double h, double w);
13    //MOVE FOR RECT
14    void move(double dX, double dY);
15    void initBox();
16 private:
17    void moveBox(double dX = 0, double dY = 0);
18 };
19 #endif
```

Before declaring the 'Rectangle' class we must first include the base class 'Form' (see the line number 4). Then, on the line 6 we start declaring the 'Rectangle' class that derives from 'Form'. The private methods are two double type variables 'width' and 'height' of a rectangle. In the private section we also have a function called 'moveBox' that changes the position of the bounding box for this rectangle. Since we don't need this function outside of class, we placed this function in the private section of this class. The public methods are an overloaded default constructor, a second overloaded constructor that initializes the private variables 'width' and 'height', an overloaded method called 'move' (overloads the 'move' method in the Form class), and a method 'initBox' that initializes the 'Box' type object for this class.

Figure 12: Rectangle class Implementation

3.8 Implementation of Rectangle class

Listing 12: My Caption

```
#include "Rectangle.h"

Rectangle::Rectangle() : width(0.0), height(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::initBox() {
   this->box.setXMax(width / 2);
   this->box.setXMin(-width / 2);
   this->box.setYMax(height / 2);
   this->box.setYMin(-height / 2);
}
```

The first overloaded constructor 'Rectangle()' zero-initializes the width and height of the rectangle and all variables of the 'Box' type object. The second constructor 'Rectangle(double h, double w)' takes the parameters and initializes the variables 'height' and 'width'. The method 'initBox' takes into account that the bounding box for the rectangle is different from any other form and it initializes the variables 'xMin', 'xMax', 'yMin' and 'yMax' of the 'box' object in this class. The overloaded method 'move' calls the 'move' function of the 'Form' class and changes the position of the rectangle according to the two parameters 'dX' and 'dY'. This method also calls the private 'moveBox' method to update the values of the 'box' object. This makes the bounding box follow the rectangle and it updates the bounding box automatically.

Figure 13: Main class Implementation

3.9 Implementation of the Main Class

Listing 13: My Caption

```
#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));
```

```
50        rect1->getBoxRef().print();
51
52        cout << "Move Rectangle in X direction for: ";
53        cin >> movX;
54        cout << "Move Rectangle in Y direction for: ";
55        cin >> movY;
56
57        rect1->move(movX, movY);
58        cout << "After Move is called: " << endl;
59        rect1->getBoxRef().print();
60     }
61
62     Circle *circle2 = NULL;
63     Rectangle *rect2 = NULL;
64     cout << "Enter second form (rectangle or circle): ";
65     cin >> prompt;
66     if (inputIsCircle(prompt)) {
67        circle2 = circleCreator(inputIsCircle(prompt));
68        circle2->getBoxRef().print();
69
70        cout << "Move Circle in X direction for: ";
71        cin >> movX;
72        cout << "Move Circle in Y direction for: ";
73        cin >> movY;
74
75        circle2->move(movX, movY);
76        cout << "After Move is called: " << endl;
77        circle2->getBoxRef().print();
78     } else if (inputIsRect(prompt)) {
79        rect2 = rectCreator(inputIsRect(prompt));
80        rect2->getBoxRef().print();
81
82        cout << "Move Rectangle in X direction for: ";
83        cin >> movX;
84        cout << "Move Rectangle in Y direction for: ";
85        cin >> movY;
86
87        rect2->move(movX, movY);
88        cout << "After Move is called: " << endl;
89        rect2->getBoxRef().print();
90     }
91
92     //ADD BOUNDING BOXES AND PRODUCE NEW ONE AS SUM
93     Box boundingBox;
94
95     cout << "Bounding Box: " << endl;
96     boundingBox = addBoxes(circle1, circle2, rect1, rect2);
97     if (!(boundingBox.getXMax() == 0.0 && boundingBox.getXMin() == 0.0 &&
           boundingBox.getYMin() == 0.0 && boundingBox.getYMax() == 0.0)) {
98        boundingBox.print();
99     }
100
101    cout << "_______________________________________" << endl;
102
103    delete circle1, rect1, circle2, rect2;
104
105    return 0;
```

```
106 }
107
108 Box addBoxes(Circle* c1, Circle* c2, Rectangle* r1, Rectangle* r2) {
109
110    Box result;
111    if (c1 == NULL && c2 == NULL) {
112       result = r1->getBoxRef() + r2->getBoxRef();
113       return result;
114    } else if (c1 == NULL && r2 == NULL) {
115       result = r1->getBoxRef()+ c2->getBoxRef();
116       return result;
117    } else if (r1 == NULL && c2 == NULL) {
118       result = c1->getBoxRef()+ r2->getBoxRef();
119       return result;
120    } else if (r1 == NULL && r2 == NULL) {
121       result = c1->getBoxRef() + c2->getBoxRef();
122       return result;
123    }
124
125    return result;
126 }
127
128 bool inputIsCircle(string prompt) {
129    string circle = "circle";
130    bool result = false;
131    if (prompt.compare(circle) == 0) {
132       result = true;
133    } else {
134       return result;
135    }
136    return result;
137 }
138
139 bool inputIsRect(string prompt) {
140    string rect = "rectangle";
141    bool result = false;
142    if (prompt.compare(rect) == 0) {
143       result = true;
144    } else {
145       return result;
146    }
147
148    return result;
149 }
150 Circle* circleCreator(bool isTrue) {
151    if (isTrue) {
152       double rad;
153
154       cout << "Enter radius: ";
155       cin >> rad;
156       Circle *circle = new Circle(rad);
157       circle->initBox();
158       return circle;
159    } else {
160       return NULL;
161    }
162 }
```

```
163 Rectangle* rectCreator(bool isTrue) {
164    if (isTrue) {
165       double h, w;
166       cout << "Enter height: ";
167       cin >> h;
168       cout << "Enter width: ";
169       cin >> w;
170       Rectangle *rect = new Rectangle(h, w);
171       rect->initBox();
172       return rect;
173    } else {
174       return NULL;
175    }
176 }
```

Now, in the main function we implement all these functions. First, we included the 'iostream' directory (line 1) and the 'string' directory from the standard library (line 2). We also need the 'Circle' and 'Rectangle' class so we included it (lines 3 and 4). To make our lives more convenient we used the using-directive for the 'std' namespace (line 6). On the lines 9, 12, 15, 18 and 19 we declare prototype functions that all have a different purpose. These are discussed briefly bellow. On the line 21 we start with the 'main' function. First, we declare two double type variables 'movX' and 'movY' that will be initialized depending on the user input and, later, used as arguments for the 'move' function of each form. The declared string 'prompt' on the line 26 is used for deciding which form to create (either a circle or a rectangle). Lines 30 and 31 declare and initialize with NULL a circle and a rectangle object. Depending on the user input (either 'circle' or 'rectangle') only one form of these two types will be initialized (i.e. 'circle1/2' or 'rect1/2') The creation is implemented in the 'circleCreator' and 'rectCreator' functions. The values of the bounding box are also displayed through the use of the 'print' method of the 'Box' class. The user will be also asked to move the form in the X and Y direction. The input is stored in the 'movX' and 'movY' variables respectively and then passed to the 'move' function. After calling the 'move' function, the 'print' method of the 'Box' class is used again to display the variables of the bounding box for this form. The whole process repeats for the second form. Then, on the line 93, we declare a new 'Box' type object called 'boundingBox'. This object is used to add the bboxes of the created forms. This is achieved through the use of 'addBoxes' function, a function that finds suitable values for the new, larger bounding box. Keep in mind that if the bounding boxes of the two created forms don't collide, then we don't need to find a new bounding box since they should stay separated from each other. On the line 103 we deallocate the memory used by 'Circle' and 'Rectangle' objects.