Frankfurt University of Applied Sciences
Fachbereich 2: Informatik und Ingenieurwissenschaften

Probeklausur
Objektorientierte Programmierung
Grundlagen

Prof. Dr. Peter Thoma, 03.07.2019

Name : _____

Vorname : _____

Matrikelnummer : _____

Hiermit bestätige ich, dass ich mich gesund fühle und in der Lage bin, an der Prüfung teilzunehmen.

Unterschrift: _____

Aufgabe	**1**	**2**	**3**	**4**	**Summe**	**Note**
Maximale Punkte	45	32	35	70	182	
Erreichte Punkte						

Zugelassene Hilfsmittel: (darüber hinaus sind keine weiteren Hilfsmittel zugelassen.**)**

- Die Vorlesungsfolien im PDF-Format befinden sich auf den Klausurrechnern unter ~/vorlagen.

Wichtige Hinweise:

1. Schreiben Sie die Lösungen auf die vorbereiteten Blätter. Eigenes Papier darf nicht verwendet werden. Bei Bedarf können Sie weitere Blätter erhalten. Beschriften Sie alle Blätter mit Ihrem Namen und Ihrer Matrikelnummer.
2. Legen Sie bitte Ihren Lichtbildausweis und Ihren Studentenausweis bereit.
3. Die Bearbeitungszeit beträgt 120 Minuten.
4. Mit Bleistift oder Rotstift geschriebene Ergebnisse werden nicht gewertet.
5. **Kopieren Sie die Lösungen zu den Programmieraufgaben vor Ende der Bearbeitungszeit in das Verzeichnis ~/ergebnisse.**
6. Schalten Sie Ihre Mobiltelefone aus und verstauen Sie diese in einer Tasche.
7. Jeder Versuch der Zusammenarbeit mit anderen Personen muss als Täuschungsversuch gewertet werden und führt zum Ausschluss von der Klausur.

Viel Erfolg bei der Klausur!

Name: _____, Vorname: _____, Matrikelnummer: _____

Aufgabe 1: Verständnisfragen

Kreuzen Sie bei den folgenden Fragen alle richtigen Antworten an (es können jeweils mehrere Antworten richtig sein). Für jedes richtig, bzw. nicht richtig gesetzte Kreuz erhalten Sie 0,25 Punkte. Zusätzlich gibt es einen weiteren Punkt pro Frage, wenn Sie diese komplett richtig beantwortet haben. Für falsche Ankreuzungen gibt es keine Abzüge.

1) Welche der folgenden Aussagen über Datentypen sind richtig?

- [X] `int` ist ein fundamentaler Datentyp in C++.
- [X] `bool` ist ein fundamentaler Datentyp in C++.
- [ ] `string` ist ein fundamentaler Datentyp in C++.
- [X] Eine Variable des Typs `int` kann positive und negative Werte annehmen.
- [X] Der C++ Standard schreibt einen festen Wertebereich für jeden Datentypen vor. Alle Systeme müssen sich genau an diesen Wertebereich halten.
- [ ] Eine Variable des Typs `long int` kann nur positive Werte annehmen.

2) Bei der Übergabe von Objekten in Argumentlisten von Funktionen oder Methoden gilt:

- [X] Bei einer „Call by Value“ Übergabe wird das Objekt kopiert.
- [ ] Bei einer Übergabe als Referenz wird das Objekt kopiert.
- [X] Bei einer Übergabe als Referenz haben Modifikationen des Objektes in der aufgerufenen Funktion immer Auswirkungen auf das Objekt der aufrufenden Funktion.
- [ ] Bei einer Übergabe als Referenz kann das Objekt innerhalb der aufgerufenen Funktion immer modifiziert werden.
- [ ] Bei einer Übergabe als Zeiger kann das Objekt innerhalb der aufgerufenen Funktion immer modifiziert werden.

3) Für die C++ Klasse `string` gilt folgendes:

- [X] Die Klasse `string` ist Bestandteil der Standardbibliothek.
- [ ] Bei Verwendung der Klasse `string` ist die zusätzliche Benutzung von `char*` nicht erlaubt.
- [X] Die Klasse `string` bietet Funktionalitäten an, um Zeichenketten, welche als `char*` oder `const char*` gespeichert sind in `string` Objekte zu konvertieren.
- [ ] Die Klasse `string` implementiert den Zuweisungsoperator und den Kopierkonstruktor.
- [ ] Die Verwendung von Objekten des Typs `string` als Klassenattribut erfordert zwingend die Definition eines speziellen Kopierkonstruktors für die Klasse.

Name: _____, Vorname: _____, Matrikelnummer: _____

4) Für die Variable `i` gilt bei Deklaration innerhalb einer `for` Anweisung, d.h.

```
for (int i=0; i<10; i++)
```

- ☐ Die Variable ist ab diesem Zeitpunkt für den Rest des aktuellen Gültigkeitsbereiches, in dem die Schleifenanweisung steht, bekannt.
- ☒ Die Variable existiert nur innerhalb des Anweisungsteils der Schleife.
- ☒ Die Variable überdeckt im Anweisungsteil der Schleife gleichnamige Klassenattribute oder Variablen, die in einem übergeordneten Gültigkeitsbereich definiert sind.
- ☐ Falls in einem übergeordneten Gültigkeitsbereich bereits eine gleichnamige Variable definiert ist, bricht der Kompiliervorgang mit einer Fehlermeldung ab.

5) Für Namen von Klassen, Methoden und Variablen gilt in C++:

- ☒ Es wird zwischen Groß- und Kleinschreibung unterschieden
- ☒ Namen dürfen keine Leerzeichen beinhalten
- ☐ Wenn ein Name Leerzeichen beinhalten, dann muss der Name in Anführungsstriche gesetzt werden.
- ☒ Die Namen von Klassen müssen innerhalb ihres Gültigkeitsbereiches eindeutig sein.
- ☐ Die Verwendung von Umlauten in Namen ist zulässig.
- ☐ Namen dürfen Zahlen an beliebiger Stelle beinhalten.
- ☒ Namen dürfen Zahlen beinhalten, allerdings nicht an erster Stelle.
- ☒ Ein Unterstrich _ ist an beliebiger Stelle in Namen zulässig.
- ☒ Namen sollten aussagekräftig gewählt werden, um die Lesbarkeit des Programmes zu unterstützen.
- ☐ Namen sollten möglichst kurzgehalten werden, um Speicherplatz zu sparen.

6) Für die dynamische Speicherverwaltung verwendet man in C++ folgende Kombinationen:

- ☒ `new` / `delete`
- ☐ `malloc` / `free`
- ☐ `new` / `delete []`
- ☒ `new []` / `delete []`

Name: _____, Vorname: _____, Matrikelnummer: _____

7) Für die Methoden von Klassen treffen folgende Aussagen zu:

- ☐ Der Destruktor kann, wie der Konstruktor auch überladen werden.
- ☐ Jeder Methodenname muss eindeutig sein.
- ☒ Es kann mehrere Methoden gleichen Namens geben, die sich durch Typ und Anzahl ihrer Argumente unterscheiden.
- ☐ Zwei Methoden können sich ausschließlich im Typ ihres Rückgabewertes unterscheiden.

8) Innerhalb einer als „`const`“ deklarierten Methode gilt folgendes:

- ☐ Lokalen Variablen dürfen nicht verändert werden.
- ☐ Globale Variablen dürfen nicht verändert werden.
- ☒ Klassenattribute dürfen nicht verändert werden.
- ☐ Es können beliebige andere Methoden der Klasse aufgerufen werden.
- ☒ Es dürfen nur ebenfalls als „`const`“ deklarierte Methoden der Klasse aufgerufen werden.

9) In der objektorientierten Programmierung gilt:

- ☐ Die Begriffe Klassen und Objekte sind identisch.
- ☒ Ein Objekt ist eine Instanz einer Klasse.
- ☒ Es können beliebig viele Objekte einer Klasse instanziiert werden.
- ☐ Es kann von jeder Klasse immer nur genau ein Objekt instanziiert werden.

10) Bei der Vererbung gelten folgende Regeln:

- ☐ Es darf nur von genau einer Klasse abgeleitet werden.
- ☐ Es darf in der abgeleiteten Klasse keine Methoden geben, die bereits mit identischer Signatur in der Basisklasse deklariert wurden.
- ☒ Es darf in der abgeleiteten Klasse Methoden geben, die bereits mit identischer Signatur in der Basisklasse deklariert wurden.
- ☐ Es darf in der abgeleiteten Klasse Methoden geben, die bereits mit identischer Signatur in der Basisklasse deklariert wurden. Hierbei ist allerdings zwingend das Schlüsselwort `virtual` voranzustellen ist.
- ☒ Bei Mehrfach-Vererbungen sind gleiche Namen von Methoden und Klassenattributen in verschiedenen Basisklassen zulässig.

Name: _____, Vorname: _____, Matrikelnummer: _____

11) Bei der Instanziierung von Objekten einer abgeleiteten Klasse sind folgende Aussage zutreffend:

- ☐ Es wird zuerst ein Konstruktor der abgeleiteten Klasse und dann einer der Basisklasse aufgerufen.
- ☒ Es wird zuerst ein Konstruktor der Basisklasse und dann einer der abgeleiteten Klasse aufgerufen.
- ☐ Es wird immer der Standard-Konstruktor der Basisklasse aufgerufen.
- ☐ Es wird kein Konstruktor der Basisklasse automatisch aufgerufen.

12) Bei Verwendung des Schlüsselwortes `virtual` für eine Methode `foo` in einer Basisklasse gilt für den Methodenaufruf über einen Zeiger auf die Basisklasse (d.h. `basisklasse->foo()`) folgendes:

- ☒ Es wird im Falle einer Überschreibung der Methode durch abgeleitete Klassen immer die überschreibende Methode des aktuellen Objektes aufgerufen.
- ☐ Es wird immer die Methode der Basisklasse aufgerufen.
- ☐ Es wird auch ohne Verwendung des Schlüsselwortes `virtual` immer die überschreibende Methode des aktuellen Objektes aufgerufen.

13) Für Methoden und Klassenattribute, die in einem `public` Bereich der Klassendeklaration deklariert sind, gilt folgendes:

- ☒ Die Methoden können innerhalb anderer Methoden der Klasse und aller eventuell abgeleiteten Klassen aufgerufen werden.
- ☐ Die Attribute sind nur innerhalb der Methoden der Klasse selbst, aber nicht in denen abgeleiteter Klassen verfügbar.
- ☒ Die Methoden können von Funktionen außerhalb der Klasse aufgerufen werden.
- ☐ Die Attribute können nicht von Funktionen außerhalb der Klasse angesprochen werden

14) Ein- und Ausgabedatenströme sind ein Konzept in C++, welches:

- ☐ Nur für Dateioperationen angewendet wird.
- ☐ Nur für Eingaben von und Ausgaben auf die Konsole relevant ist.
- ☒ Unter anderem für Dateioperationen und Konsolen Ein- und Ausgaben verwendet wird.
- ☐ Keine internen Zustände kennt.

Name: _____, Vorname: _____, Matrikelnummer: _____

15) Beim Überladen von Operatoren gilt folgendes:

- ☐ Das Objekt, für dessen Klasse der Operator deklariert ist, muss rechts vom Operator stehen.
- ☒ Operatoren, die für Objekte rechts vom Operator deklariert werden sollen, müssen als globale Operatoren deklariert werden.
- ☐ Der Vergleichsoperator kann nicht überschrieben werden.
- ☐ Der Zuweisungsoperator muss immer als globaler Operator deklariert werden.
- ☒ Der Standard-Zuweisungsoperator erzeugt eine binäre (flache) Kopie des Objektes.
- ☒ Bei Verwendung von dynamischer Speicherverwaltung für die Klassenattribute muss der Zuweisungsoperator überschrieben werden, um eine tiefe Kopie des Objektes zu erzeugen.

16) Für die unterschiedlichen Speicherbereiche eines Programmes gilt:

- ☒ Lokal deklarierte Variablen werden immer auf dem Stapelspeicher („Stack“) angelegt.
- ☒ Mit dem `new` Operator wird Speicher im Haldenspeicher („Heap“) reserviert.
- ☒ Die Größe des Haldenspeichers („Heap“) ist durch die Zeigerlänge (d.h. 32 Bit, 64 Bit) und durch den verfügbaren Speicher des Computers beschränkt.
- ☒ Die Größe des Stapelspeichers („Stack“) ist begrenzt und kann durch eine Option bei der Kompilierung des Programmes festgelegt werden.

17) Wichtige Ziele guten objektorientierten Designs sind:

- ☒ Die Wiederverwendbarkeit des Programmcodes erleichtern.
- ☐ Die Größe des ausführbaren Programmes reduzieren.
- ☒ Die Abhängigkeiten zwischen Programmteilen reduzieren.
- ☐ Die Geschwindigkeit der Programmausführung zu erhöhen.
- ☒ Die Wartbarkeit des Programmcodes zu verbessern.

Name: _____, Vorname: _____, Matrikelnummer: _____

18) Wie kann ein gutes objektorientiertes Design für ein gegebenes Problem gefunden werden?

- [X] Durch Befolgung der unter dem Begriff SOLID bekannten wichtigen Prinzipien guten Designs.
- [ ] Durch eine strikte Umsetzung bekannter Entwurfsmuster auch wenn diese nicht vollständig zu dem gegebenen Problem passen.
- [X] Durch die Auswahl passender Entwurfsmuster und deren Anpassung auf die gegebene Problemstellung.
- [ ] Durch Befolgung der Regel „hat ein“ für eine Vererbung.
- [X] Durch Befolgung der Regel „ist ein“ für eine Vererbung.

19) Welche der folgenden Aussagen über die UML sind zutreffend:

- [X] In der UML werden Klassen durch Rechtecke repräsentiert.
- [ ] In der UML werden Objekte durch Ellipsen repräsentiert.
- [X] Beziehungen zwischen Klassen werden durch Linien darstellt.
- [ ] Vererbungshierarchien zwischen Klassen werden durch einen offenen Pfeil → dargestellt.
- [ ] Der Pfeil zur Darstellung der Vererbungshierarchie geht von der Basisklasse zur abgeleiteten Klasse.

20) Welche der folgenden Aussagen über C++ Templates sind zutreffend:

- [X] In C++ können Templates sowohl für Funktionen als auch für Klassen erstellt werden.
- [X] Templates dienen der Verallgemeinerung von Datentypen.
- [ ] std::string ist eine Templateklasse.
- [X] Mit Hilfe des Templates std::list können doppelt verkette Listen verwaltet werden.
- [ ] Der Programmcode von Templates sollte wie der von Klassen auf eine Deklarationsdatei und eine Implementierungsdatei aufgeteilt werden.
- [ ] Bei der Instanziierung einer Templateklasse wird der gewünschte Datentyp in runden Klammern () angegeben.
- [X] Bei Klassentemplates können neben dem Typ noch weitere Parameter angegeben werde, die dann innerhalb der Klasse als Konstante verwendet werden können.
- [X] Die Standardbibliothek verwendet das Konzept der Iteratoren, um über die Elemente von Datencontainern zu iterieren.

Name: _____, Vorname: _____, Matrikelnummer: _____

Aufgabe 2: Code-Verständnis

Bitte betrachten Sie folgenden Programmcode einer Klasse zur Verwaltung von Fahrzeugen. Diese Klasse dient als Basisklasse für die Spezialisierung verschiedener Fahrzeugarten.

Die Deklaration der Klasse ist in der Datei *fahrzeug.h* wie folgt gegeben:

```
#include <string>

class Fahrzeug
{
public:
      Fahrzeug();
      Fahrzeug(const char *n);
      ~Fahrzeug();

      void setName(const char *n);
      const char *getName() const;
private:
      char *name;
};
```

Die Implementierung der Klasse in der Datei *fahrzeug.cpp* lautet wie folgt:

```
#include "Fahrzeug.h"
#include <string.h>

Fahrzeug::Fahrzeug()
{
}

Fahrzeug::Fahrzeug(const char *n)
{
      setName(n);
}

Fahrzeug::~Fahrzeug()
{
      if (name != nullptr) delete [] name;  name = nullptr;
}

void Fahrzeug::setName(const char *n)
{
      if (name != nullptr) delete name;
      name = new char[strlen(n)+1];
      strcpy(name, n);
}

const char *Fahrzeug::getName() const
{
      return name;
}
```

Name: _____, Vorname: _____, Matrikelnummer: _____

1) Beschreiben Sie kurz die Funktion der Methode `setName` und gehen Sie dabei auf eventuelle Fehler in der Implementierung ein:

The method 'setName()' gives objects of other classes the ability to set the private member 'char *name' to a new value.
The given implementation is problematic, because it does not handle a possible dangling pointer to freed memory after 'delete name;' and also doesn't account for a possible nullptr that could cause strlen to crash.

2) Was könnte passieren, wenn die Zeile

```
if (name != nullptr) delete name;
```

in der Implementierung der Methode `setName` entfallen würde?

Leaving the line out leads to a memory leak, because the pointer of the member 'name' would be changed to the newly allocated memory in the following line without clearing the old memory. The old memory cannot be reached anymore afterwards.
Better implementation:

```
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);
        }
}
```

3) Sind Deklaration und Implementierung der Konstruktoren und Destruktoren korrekt? Falls nein, was sollte korrigiert werden?

The provided implementation of the standard constructor does not initialize the private attribute 'char *name', so it is not guaranteed, that it is a nullptr initially.
The overloaded constructor does not need to handle this problem, since it uses the setName() method, assuming it is implemented correctly.
The destructor is perfectly fine, but 'name = nullptr;' is not necessary, because the pointer belongs to the object that is being destructed and does not exist anymore after the execution of the destructor.

Name: _____, Vorname: _____, Matrikelnummer: _____

4) Beschreiben Sie, warum Sie für diese Klasse einen Kopier-Konstruktor und einen Zuweisungsoperator benötigen.

We need to implement the copy constructor and overwrite the assignment operator, because the class 'Fahrzeug' contains a member variable which is a pointer to memory outside of the object, so we have to manage the memory allocation manually.

5) Beschreiben Sie (handschriftlich) eine mögliche Implementierung für den Zuweisungsoperator:

```
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;
}
```

6) Wie könnte der Programmcode der Klasse unter Verwendung der Standardbibliothek vereinfacht werden? Würden in diesem Fall noch ein spezielle Kopierkonstruktor und ein spezieller Zuweisungsoperator benötigt?

We could use std::string instead of *char. std::string handles memory allocation internally, so we wouldn't need to have a special copy constructor or overwrite the assignment operator.

Name: _____, Vorname: _____, Matrikelnummer: _____

Aufgabe 3: Programmieraufgabe „Buch“

Wichtige Hinweise:

- Schreiben Sie Ihren Namen und Ihre Matrikelnummer als Kommentar an den Anfang jeder Datei.
- Trennen Sie die Deklaration und Implementierung der Klassen jeweils in eine Headerdatei *Klassenname.h* und eine Implementierungsdatei *Klassenname.cpp*.
- Achten Sie darauf, dass das Programm korrekt kompiliert.
- **Nach Bearbeitung der Aufgabe kopieren Sie entweder das gesamte Eclipse Projekt oder alle Quellcodedateien (Klassendateien, Hauptprogramm) Ihrer Lösung in ein Unterverzeichnis „Aufgabe 3“, welches Sie in ~/ergebnisse erstellen.** Hierfür können Sie entweder das Projekt mit der Funktion File→Export→General→Archive File, Auswahl Ihres Projektes und Angabe eines „To archive file“ in dem Ergebnisverzeichnis verwenden. Alternativ können Sie Ihre Dateien einzeln mit Save As in das Verzeichnis speichern.

1) Implementieren Sie eine Klasse „eBook“, welche zur Verwaltung eines eBooks verwendet werden kann. Das Buch wird dabei durch eine Zeichenkette für den Titel und eine Zeichenkette für den Inhalt repräsentiert (jeweils `std::string`). Ermöglichen Sie lesenden und schreibenden Zugriff auf den Titel und den Inhalt des Buches, ohne jedoch die Klassenattribute direkt von außerhalb der Klasse ansprechbar zu machen.

2) Implementieren Sie eine Methode `print`, welche den Titel und den Inhalt des Buches auf `std::cout` ausgibt.

3) Fügen Sie einen Konstruktor hinzu, der das Setzen des Titels und des Inhaltes ermöglicht.

4) Überladen Sie den Ausgabeoperator << für Ihre Klasse, wobei analog zur Methode `print` zunächst der Titel und danach der Inhalt des Buches in den Ausgabedatenstrom geschrieben werden.

5) Schreiben Sie ein Hauptprogramm `main`, welches ein Objekt Ihrer Klasse mit einem beliebigen von Ihnen gewählten Titel und Inhalt instanziiert. Danach geben Sie das Buch mit Hilfe der `print` Methode auf die Konsole aus. Abschließend öffnen Sie eine Datei namens *Buch.txt* und schreiben Titel und Inhalt des Buches unter Verwendung des Ausgabeoperators << in die Datei.

Name: _____, Vorname: _____, Matrikelnummer: _____

Aufgabe 4: Programmieraufgabe „Formen“

Wichtige Hinweise:

- Schreiben Sie Ihren Namen und Ihre Matrikelnummer als Kommentar an den Anfang jeder Datei.
- Trennen Sie die Deklaration und Implementierung der Klassen jeweils in eine Headerdatei *Klassenname.h* und eine Implementierungsdatei *Klassenname.cpp*.
- Achten Sie darauf, dass das Programm korrekt kompiliert.
- **Nach Bearbeitung der Aufgabe kopieren Sie entweder das gesamte Eclipse Projekt oder alle Quellcodedateien (Klassendateien, Hauptprogramm) Ihrer Lösung in ein Unterverzeichnis „Aufgabe 3“, welches Sie in ~/ergebnisse erstellen.** Hierfür können Sie entweder das Projekt mit der Funktion File→Export→General→Archive File, Auswahl Ihres Projektes und Angabe eines „To archive file“ in dem Ergebnisverzeichnis verwenden. Alternativ können Sie Ihre Dateien einzeln mit Save As in das Verzeichnis speichern.

1) Entwickeln Sie eine Klasse Circle, welche einen Kreis durch einen Radius „radius“ vom Typ `double` repräsentiert. Für die Erzeugung des Kreises wird ein überladener Konstruktor verwendet, bei dem der Radius als Argument angegeben wird. Darüber hinaus benötigen Sie der Einfachheit halber keinen externen Zugriff auf die Daten des Kreises.

2) Entwickeln Sie eine Klasse Rectangle, welche ein Rechteck durch eine Breite „width“ und eine Höhe „height“ – jeweils vom Typ `double` - repräsentiert. Für die Erzeugung des Rechteckes wird ein überladener Konstruktor verwendet, bei dem Breite und Höhe als Argumente angegeben werden können. Darüber hinaus benötigen Sie der Einfachheit halber keinen externen Zugriff auf die Daten der Rechtecke.

3) Nun Erstellen Sie eine Basisklasse Form, welche den Mittelpunkt der Form („xcenter“, „ycenter“) – jeweils vom Typ `double` – beschreibt. Bei er Instanziierung der Objekte wird der Mittelpunkt der Form mit (0.0, 0.0) initialisiert.

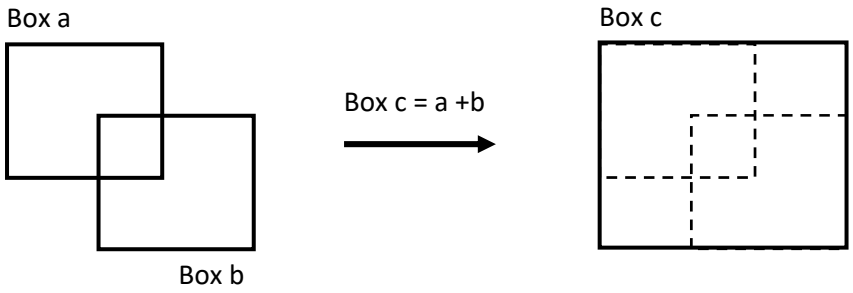
4) Leiten Sie die beiden Klassen Rectangle und Circle jeweils von der gemeinsamen Basisklasse Form ab.

5) Ergänzen Sie die Klasse Form um eine Methode „`move`“ mit zwei Argumenten vom Typ `double` („dx“, „dy“). Bei Aufruf dieser Methode wird der Mittelpunkt der Form um dx in x-Richtung und um dy in y-Richtung verschoben.

6) Nun möchten Sie die Abmessungen der Form durch eine sogenannte „Bounding-Box“ beschreiben. Hierzu erstellen Sie zunächst eine weitere Klasse „Box“ mit den Klassenattributen „xmin“, „xmax“, „ymin“ und „ymax“ – alle vom Typ `double`. Initialisieren Sie diese Variablen bei der Instanziierung mit einem Standard-Konstruktor mit dem Wert 0.0. Schützen Sie die Variablen vor direktem Zugriff von außerhalb der Klasse und führen Sie entsprechende Methoden zum Lesen und Setzen der Werte ein.

Name: _____, Vorname: _____, Matrikelnummer: _____

7) Überladen Sie den `Operator+` für die Klasse Box. Dabei werden die Boxen der beiden Summanden so „addiert“, dass eine neue Box entsteht, die die beiden ursprünglichen Boxen einschließt, wie in folgendem Bild veranschaulicht:



Hinweis: Die Standardbibliothek stellt eine Funktion `std::min(a,b)` zur Verfügung, welche das Minimum der beiden Parameter a und b zurückliefert. Analog gibt es eine Funktion `std::max(a,b)`. Beide Funktionen sind in der Headerdatei `<algorithm>` deklariert.

8) Nun ergänzen Sie die Klasse Box um eine Methode `print`, welche die Abmessungen der Bounding-Box auf die Konsole ausgibt.

9) Entsprechend der Beziehung „Die Form hat eine Bounding-Box“ fügen Sie nun der Klasse Form ein Klassenattribut vom Typ Box hinzu. Führen Sie darüber hinaus der Klasse Form eine öffentliche Methode hinzu, die eine nicht veränderbare Referenz auf die Bounding-Box liefert. Das Attribut soll von einem direkten Zugriff von außerhalb der Klasse geschützt werden, soll aber in abgeleiteten Klassen direkt verwendet werden können.

10) Bei der Erzeugung der Rechtecke und Kreise fügen Sie nun entsprechenden Programmcode hinzu, der die Bounding-Box gemäß den Abmessungen der jeweiligen Form setzt.

11) Für die Methode `move` ergänzen Sie ebenfalls entsprechenden Programmcode, der die Bounding-Box entsprechend der Verschiebung anpasst.

12) Nun schreiben Sie ein Hauptprogramm, mit dessen Hilfe der Anwender nacheinander zwei Formen erzeugen kann. Für jede dieser Formen kann ausgewählt werden, ob ein Rechteck oder ein Kreis erzeugt wird. Je nach Wahl werden danach die Parameter der entsprechenden Form abgefragt und vom Anwender über die Konsole eingegeben (d.h. Breite und Höhe bzw. Radius). Dabei wird der Einfachheit halber angenommen, dass die Eingaben korrekt sind, d.h. es sind keine Überprüfungen der Eingaben notwendig. Bei Bedarf können Sie Teile der Funktionalität in Unterprogramme auslagern.

13) In einem weiteren Schritt werden die Bounding-Boxen der beiden Formen abgefragt und addiert, so dass sich eine Bounding-Box für die beiden Formen gemeinsam ergeben. Geben Sie danach die resultierende Bounding-Box mit ihrer `print` Methode auf die Konsole aus.

14) Nun werden nacheinander vom Anwender über die Konsole Verschiebungen für die beiden zuvor eingegebenen Formen in der Form „Verschiebung Form 1 in x Richtung“, etc. abgefragt. Basierend auf diesen Daten verschieben Sie die jeweilige Form mit Hilfe der `move` Methode.

15) Abschließend ermitteln Sie wieder die gemeinsame Bounding-Box der beiden Formen und geben Sie diese erneut auf die Konsole aus.

Klausur Objektorientierte Programmierung Grundlagen

Name: _____, Vorname: _____, Matrikelnummer: _____

Zusatzblatt zu Aufgabe _____

Dokument anlegen:
Titelblatt
alle Namen
Inhaltsverzeichnis, Abb. verzeichnis
Tabellenverzeichnis

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

```
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

```
1  #ifndef _EBOOK_H_
2  #define _EBOOK_H_
3
4  #include <string>
5  #include <iostream>
6
7  using namespace std;
8
9  class EBook {
10 private:
11    string title, content;
12 public:
13    EBook();
14    EBook(string title, string content);
15    void setTitle(string title);
16    string getTitle() const;
17    void setContent(string content);
18    string getContent() const;
19    void print() const;
20    friend ostream &operator<<(ostream &output, const EBook &book);
21 };
22
23 #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

```
1  #include "eBook.h"
2  #include <iostream>
3
4  using namespace std;
5
6  EBook::EBook() : title(""), content("") {};
7
8  EBook::EBook(string title, string content) : title(title), content(content) {};
9
10 void EBook::setTitle(string title) {
11    if (title != "") {
12       this->title = title;
13    } else {
14       cout << "Title not set!" << endl;
15    }
16 }
17 string EBook::getTitle() const {
18    return this->title;
19 }
20 void EBook::setContent(string content) {
21    if (content != "") {
22       this->content = content;
23    } else {
24       cout << "Content not set!" << endl;
25    }
26 }
27 string EBook::getContent() const {
28    return this->content;
29 }
30
31 void EBook::print() const {
32    cout << "Title: " << this->title << '\n';
33    cout << "Content: " << this->content << '\n';
34 }
35
36 ostream & operator<<(ostream &output, const EBook &book) {
37    book.print();
38    return output;
39 }
```

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 Form class

2.3 Main class

```
#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

```
1  #ifndef _FORM_H_
2  #define _FORM_H_
3
4  #include "Box.h"
5
6  class Form {
7  private:
8     double xCenter, yCenter;
9  protected:
10    Box box;
11 public:
12    Form();
13    void move(double dX, double dY);
14    Box &getBoxRef();
15 };
16 #endif
```

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.

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

```
1  #include "Form.h"
2
3  Form::Form() : xCenter(0.0), yCenter(0.0) {
4
5  }
6
7  void Form::move(double dX, double dY) {
8     this->xCenter += dX;
9     this->yCenter += dY;
10 }
11
12 Box & Form::getBoxRef() {
13    return box;
14 }
```

3.3 Box Headerfile

```
1  #ifndef _BOX_H_
2  #define _BOX_H_
3
4  class Box {
5  private:
6     double xMin, xMax, yMin, yMax;
7  public:
8
9     Box();
10    double getXMin() const;
11    double getXMax() const;
12    double getYMin() const;
13    double getYMax() const;
14    void setXMax(double val);
15    void setXMin(double val);
16    void setYMin(double val);
17    void setYMax(double val);
18    friend Box operator+(Box left, Box right);
19    void print() const;
20 };
21 #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.

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

```
#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;
```

```
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

```
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 setUpBox();
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 'setUp' 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

```
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::setUpBox() {
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 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.

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

```
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
```

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.

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

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

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

```
#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();

```

```
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->setUpBox();
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->setUpBox();
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.