Praktikum
Programmierung 2

Konzeption:
Prof. Dr. Oliver Henkel

Durchführung:
Prof. Dr. Frank Thiesing
Ralf Koller
Malte Glüsenkamp

Wintersemester 2020/21

Praktikumsregeln

§1 Ziel des Praktikums ist, Programmierfertigkeiten zu erlernen und durch die kontinuierliche Bearbeitung der Praktikumsaufgaben (§5) nachzuweisen. Im Praktikum herrscht deshalb Anwesenheits- und Aktivitätspflicht (praktikumsnahe Aktivität, ansonsten zählt der Termin als ein Fehltermin).

§2 Bis zu 3 Fehltermine werden toleriert und müssen nicht begründet werden.

§3 Zwischentests: Es werden 2-3 Zwischentests durchgeführt. Die Zwischentests bestehen aus einer Programmieraufgabe, die an je einem Termin im Praktikum vollständig bearbeitet und abgegeben wird. Die Termine werden rechtzeitig (d.h. mindestens eine Woche im voraus) bekannt gegeben. Die Aufgaben orientieren sich an den Praktikumsaufgaben, bzw. den Anforderungen in der Klausur. ⟶ Wer sich intensiv (und im wesentlichen erfolgreich) mit den Praktikumsaufgaben beschäftigt hat, hat gute Karten bei den Zwischentests

§4 Das Praktikum ist sicher bestanden, wenn folgende Bedingungen erfüllt sind:

- Anwesenheit und Aktivität nach §1 und §2 erfüllt
- Es wurde mindestens die Hälfte der Gesamtpunktzahl in den Zwischentests (§3) erreicht.

§5 Das Praktikum wird nicht benotet und kann beliebig oft wiederholt werden.

§6 Zur Vorbereitung auf die Zwischentests und die Klausur dienen die hier vorliegenden Aufgaben. Die Praktikumsbetreuer stehen zur Beratung bei der Bearbeitung der Aufgaben zur Verfügung.

- Die Aufgaben sind thematisch gruppiert und so ausgelegt, dass sie in dem angegebenen Zeitraum bearbeitet werden können. Mit [WICHTIG] markierte Aufgaben sollten unbedingt bearbeitet werden.
- Alle notwendigen Daten zur erfolgreichen Bearbeitung der Aufgaben sind in der jeweiligen Aufgabenstellung aufgeführt oder in Form zusätzlicher Dateien den Aufgaben beigefügt. Eine Verwendung von nicht in den Aufgabenstellungen enthaltenen Hilfsmitteln ist nur dann eine zulässige Aufgabenbearbeitung, wenn diese Hilfsmittel detailliert dargelegt, verstanden und in vollem Umfang erklärt werden können.

Kodierregeln

Dokumentation

- Jedes Programm enthält einen Kopf mit Programmnamen und Kurzbeschreibung, Autor, Datum und ggf. einer Liste der ToDos (noch zu erledigender Änderungen, Fehlerbeseitigungen, zu entfernender Warnungen etc.):

```
/*
 * ------------------------------------------------------------
 *
 * Aufgabe: xxx.cpp
 * Autor:
 *
 * kurzer Beschreibungstext
 *
 * ggf. Testfaelle/Bemerkungen/ToDos
 *
 *
 * Praktikum Programmierung 2, HS Osnabrueck
 * ------------------------------------------------------------
 */
```

- Jede Funktion/Methode erhält einen Kopf mit Namen, kurzer Beschreibung, Parameterliste und Rückgabewerte:

```
/*
 * Funktion: xxx
 * kurzer Beschreibungstext
 *
 * in|out param1 : Typ -- Beschreibung
 * in|out param2 : Typ -- Beschreibung
 * ggf. noch eine Zeile fuer Ausnahmen:
 * throws        : Typ -- Beschreibung
 *
 * return        : Typ -- Beschreibung
 *
 */
```

- Zwischen sinnvollen Abschnitten des Programms werden Kommentare zur Trennung eingefügt.
  Beispiel: Eingabe, Berechnung, Ausgabe

- Der Quellcode enthält auch in Kommentaren keine Umlaute
- Der Quellcode soll einheitlich formatiert sein. Insbesondere sind Anweisungsblöcke zwischen öffnender und schließender geschweifter Klammer geeignet einzurücken.

Variablen / Klassenattribute

- Variablen werden nicht für unterschiedliche Zwecke im Programm benutzt. Für jeden einzelnen Zweck ist eine eigene Variable zu definieren.
- Alle verwendeten Namen sind sprechend. Abkürzungen dürfen nur verwendet werden, wenn sie für den Programmierzweck eine eigenständige Bedeutung haben (z.B. physikalische Einheiten, Bezeichner in mathematischen Ausdrücken...)
- Variablen-, Attributs-, Methoden- und Funktionsnamen beginnen mit einem Kleinbuchstaben. Bei aus mehreren Wörtern zusammengesetzten Namen (nicht: zusammengesetzte Wörter wie Startpunkt, Anfangswert) werden ab dem zweiten Wort die ersten Buchstaben jeweils groß geschrieben (Beispiel: `uhrGestartet, zaehlerPunkte`)
- Nur Schleifenvariablen oder Variablen mit einer sehr kurzen Lebensspanne dürfen einfachere Namen haben
- Klassennamen beginnen mit einem Großbuchstaben. Für jede Klasse gibt es (in C++) eine Header- und eine Quellcode-Datei, deren Dateinamen mit dem Klassennamen übereinstimmen
- Konstanten werden groß geschrieben (ggf. durchgehend, mit Unterstrich als Worttrenner)

Inhaltsverzeichnis

Kapitel 1

Java2Cpp [3 Wochen]

1.1 Ein- und Ausgabe

Erstellen Sie in C++ folgendes Programm: Lesen sie aus der Datei `eingabe.txt` zeilenweise die Werte aus und schreiben diese formatiert in die Datei `ausgabe.txt`. Die Formatierung soll über Manipulatoren erfolgen.

eingabe.txt-Zeile: `1343 4.23423434 HalloWelt`

Die einzelnen Zeilen sollen nummeriert werden. Die Zahl soll als hexadezimale Zahl dargestellt werden. Die hexadezimale Zahl soll eine minimale Breite von vier haben und durch Nullen aufgefüllt werden. Die double-Zahl soll mit sechs Zeichen ausgegeben werden. Der String soll eine minimale Länge von 15 haben. Die fehlenden Ziffern sollen mit Minuszeichen aufgefüllt werden Die einzelnen Spalten sollen durch | getrennt werden

ausgabe.txt-Zeile: `1. | 053f | 4.23423 | ------HalloWelt`

```
eingabe.txt:
1343 4.23423434 HalloWelt
34234 4323.1234243 C++machtSpaß
65455 4.30658456 Testing
3412 1.234567 abcderfghijklmn
65535 5.3620909482 Hexadezimal
32342 3423.343423 Referenzen
9834 09.934343 yxzuerf
003234 9883.32414 ABCDERFGHIJKLM
213 3.141593239853 Beispiel

ausgabe.txt
1. | 053f | 4.23423 | ------HalloWelt
2. | 85ba | 4323.12 | ---C++machtSpaß
3. | ffaf | 4.30658 | --------Testing
4. | 0d54 | 1.23457 | abcderfghijklmn
5. | ffff | 5.36209 | ----Hexadezimal
6. | 7e56 | 3423.34 | -----Referenzen
7. | 266a | 9.93434 | --------yxzuerf
8. | 0ca2 | 9883.32 | -ABCDERFGHIJKLM
9. | 00d5 | 3.14159 | -------Beispiel
```

1.2 Funktionen

Implementieren Sie die folgenden Funktionen:

1. Schreiben Sie eine Funktion `abschneiden`, die als ersten Parameter eine float-Variable `x` (zwischen 0 und 1000) und als zweiten Parameter einen positiven int-Wert `n` (zwischen 1 und 5) übergeben bekommt. Die Funktion soll den ersten Parameter auf `n` Nachkommastellen begrenzen (durch Abschneiden der restlichen Nachkommastellen). Das Ergebnis soll in `x` stehen. Die Funktion soll ohne Nutzung von Funktionen der mathematischen Bibliothek implementiert werden.

2. Schreiben Sie eine Funktion `ggT`, die den größten gemeinsamen Teiler der beiden übergebenen nichtnegativen ganzen Zahlen berechnet. Das Ergebnis soll über den ersten übergebenen Parameter zurückgegeben werden.

1.3 Listenbearbeitung

Schreiben Sie mindestens zwei der folgenden Funktionen, die für je zwei übergebene integer-Listen

- überprüft, ob die Listen gemeinsame Werte besitzen
  (Rückgabe: Liste dieser Werte (Die Liste soll keine doppelten Elemente enthalten))
- überprüft, ob die Listen keine gemeinsamen Werte besitzen
  (Rückgabe: true/false)
- genau dann true zurückliefert, wenn alle Werte der ersten Liste genauso häufig in der zweiten Liste vorkommen und umgekehrt
- die erste Liste mit denjenigen Elementen der zweiten Liste auffüllt, die nur in der zweiten Liste vorkommen

Diese Aufgabe können Sie entweder „von Hand“ programmieren oder durch geschickten Einsatz von Containermethoden kurze Lösungen finden.

Schreiben Sie ein Hauptprogramm, das Ihre Funktionen testet. Denken Sie auch an die leere Liste.

1.4 Operatormethoden [wichtig]

Prädikate

Schreiben Sie Prädikate (mindestens je ein unäres und ein binäres)

- `istTeilerVon_n`: prüft, ob die übergebene ganze Zahl ein Teiler der im Prädikat hinterlegten Zahl n ist
- `istNahe`: prüft, ob die übergebene double-Zahl sich von der hinterlegten double Zahl um höchstens **double** `tolerance` unterscheidet. `tolerance` besitzt als Default-Wert 10^{-4} und kann im Konstruktor verändert werden.
- `istKuerzerAls`: prüft für die beiden übergebenen `string`-Objekte, ob der erste übergebene String eine kürzere Länge als der zweite hat.
- `besitztMehrWorteAls`: prüft ob der erste der übergebenen Texte (Typ: `string`) aus mehr Worten besteht als der zweite. Ein Wort ist ein nicht durch Whitespace getrennte Sequenz von mindestens 2 Zeichen.

Operatormethoden

Eine Ringliste ist ein logisch ringförmig angelegter Speicherbereich (Anwendungsfälle z.B. Ein-/Ausgabepuffer) fester Größe (Kapazität). Es gibt je einen Marker für die aktuelle Lese- und Schreibposition. Diese stellen den logischen Beginn und das logische Ende der in der Ringliste gespeicherten Datensequenz dar. Beim Beschreiben wird an der Schreibposition ein neuer Wert eingetragen und der Positionsmarker rückt eine Position vor. Beim lesenden Zugriff wird der Wert an der Leseposition ausgelesen und der Marker rückt eine Position vor (damit gilt das ausgelesene Element als aus der Ringliste „entfernt“). Das Vorrücken geschieht zyklisch, d.h. wird von der letzten Position gelesen oder geschrieben, rückt der zugehörige Marker auf die Anfangsposition. Werden mehr Schreiboperationen durchgeführt als die Kapazität des Speichers beträgt, so werden die ältesten eingetragenen Werte mit den neuen überschrieben.

Gegeben sei ein Verwendungsbeispiel einer zu schreibenden Klasse Ringliste (Datei: `RinglisteSpielerei`), die eine Ringliste für integer-Werte realisiert. Die Kapazität wird dabei im Konstruktor gesetzt und besitzt den Default-Wert 10. Die integer-Daten sollen intern in einem `vector`-Container gespeichert werden. Lese- und Schreibposition sollen anfangs mit 0 initialisiert sein. Die Klasse soll alle Standardmethoden

(Konstruktoren, Destruktoren, Zuweisungsoperator) ausprogrammiert enthalten, sowie diejenigen Methoden, sodass das Verwendungsbeispiel die Ausgabe

```
0/5 |
3/5 | 1 2 3
5/5 | 2 3 4 5 6
true
3/10 | 9 9 9
5/5 | 5 6 9 9 9
3/10 | 9 9 9
```

produziert.

Hinweise:

- Beachten Sie die zyklische Struktur, bzw. die möglichen Fälle für die Positionen der Lese- und Schreibmarken. Es ist möglich, dass der Schreibmarker den Lesemarker „überholt“ oder zyklisch an den Anfang springt und dann „vor“ der Lesemarke sitzt. Es ist ebenfalls möglich, dass die beiden Marker an derselben Position stehen: Direkt nach der Objektkonstruktion (oder nach dem Auslesen sämtlicher Elemente) sind solche Ringlisten leer, nach Auffüllen mit der Maximalzahl von Werten aber nicht.
- Der Vergleich zweier Ringlisten ist rein inhaltlich und hängt nur von den in den Ringlisten gespeicherten Werten und ihrer Anordnung ab, nicht von ihrer tatsächlichen Position im `vector`-Container.
- Beachten Sie außerdem, dass die Operation „+=“ in diesem Beispiel kein `Ringliste`-Objekt ist, die Operatormethode **operator**+= also von den Konventionen der Standardimplementation abweicht

Kapitel 2

Zeiger [3 Wochen]

2.1 Fifo [wichtig]

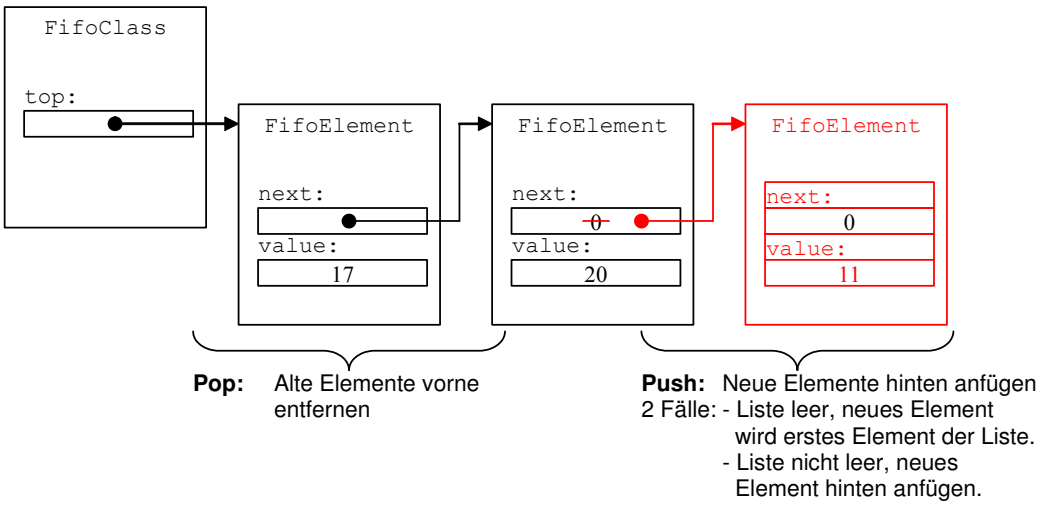
Erstellen Sie Klasse `Fifo`, die einen Fifo-Speicher für Objekte vom Typ string mit den Standardmethoden und mindestens folgenden öffentlichen Methoden realisiert:

`Fifo`()	erzeugt ein leeres Fifo
~`Fifo`()	Destruktor
`Fifo`& **operator**<<(**const** `string`&)	schreibt eine Kopie des string-Objektes in das Fifo ein ⌜überlicherweise als `push` bezeichnet⌟
`Fifo`& **operator**>>(`string`&)	liest ein string-Objekt aus dem Fifo aus ⌜überlicherweise als `pop` bezeichnet⌟
operator int () **const**	gibt den Füllstand des Fifos zurück

Ausnahmen sollen jeweils durch Werfen einer Zeichenkettenkonstante abgefangen werden. Neben unbekannten Ausnahmen sind insbesondere Ausnahmen für Fehler bei der dynamischen Speicherbeschaffung [>a] Fehler beim Öffnen von nichtexistierenden Dateien, sowie der Fifo-Unterlauf zu fangen.

Die einzelnen Elemente des Fifos sollen Objekte einer von Ihnen zu schreibenden Klasse `FifoElement` sein, die als verzeigerte Liste verkettet den Fifo im Speicher dynamisch abbilden. Nachstehendes Diagramm zeigt schematisch den Aufbau, sowie die grundlegenden Fifo-Operationen anhand eines Fifos für integer-Zahlen

[>a] `new` wirft bei Fehlern Ausnahmeobjekte vom Typ `bad_alloc`, wenn der Header `new` eingebunden ist. In so einem Fall soll die Ausnahme lokal gefangen und stattdessen eine passende Fehlermeldung weitergeworfen werden



Quelle: Prof. Dr. B. Lang, HS Osnabrück

Die Fifo-Klasse muss mit dem Testprogramm `FifoClassTest.cpp` durchgetestet werden und die Ausgabe `Ausgabe.FifoClassTest` produzieren. Dem Testprogramm entnehmen Sie ebenfalls ggf. zu erstellende öffentliche Methoden der Fifo-Klasse.

Zusatzaufgabe: Überladen Sie den <<-Operator zusätzlich so, dass er als Argument auch Fifo-Objekte (als Referenz) akzeptiert und dessen Elemente in ∗**this** einfügt.

Hinweise: Für die Bearbeitung ist die Verwendung der STL nicht erlaubt. Vielmehr soll der Fifo mit einer von Ihnen geschriebenen dynamischen Speicherverwaltung versehen werden. Achten Sie darauf, dass Kopierkonstruktor und Zuweisungsoperator tiefe Kopien erzeugen und keine Speicherlecks entstehen.

Kapitel 3

Vererbung [2 Wochen]

3.1 GeoObjekt [wichtig]

Implementieren Sie nachstehendes Klassendiagramm.

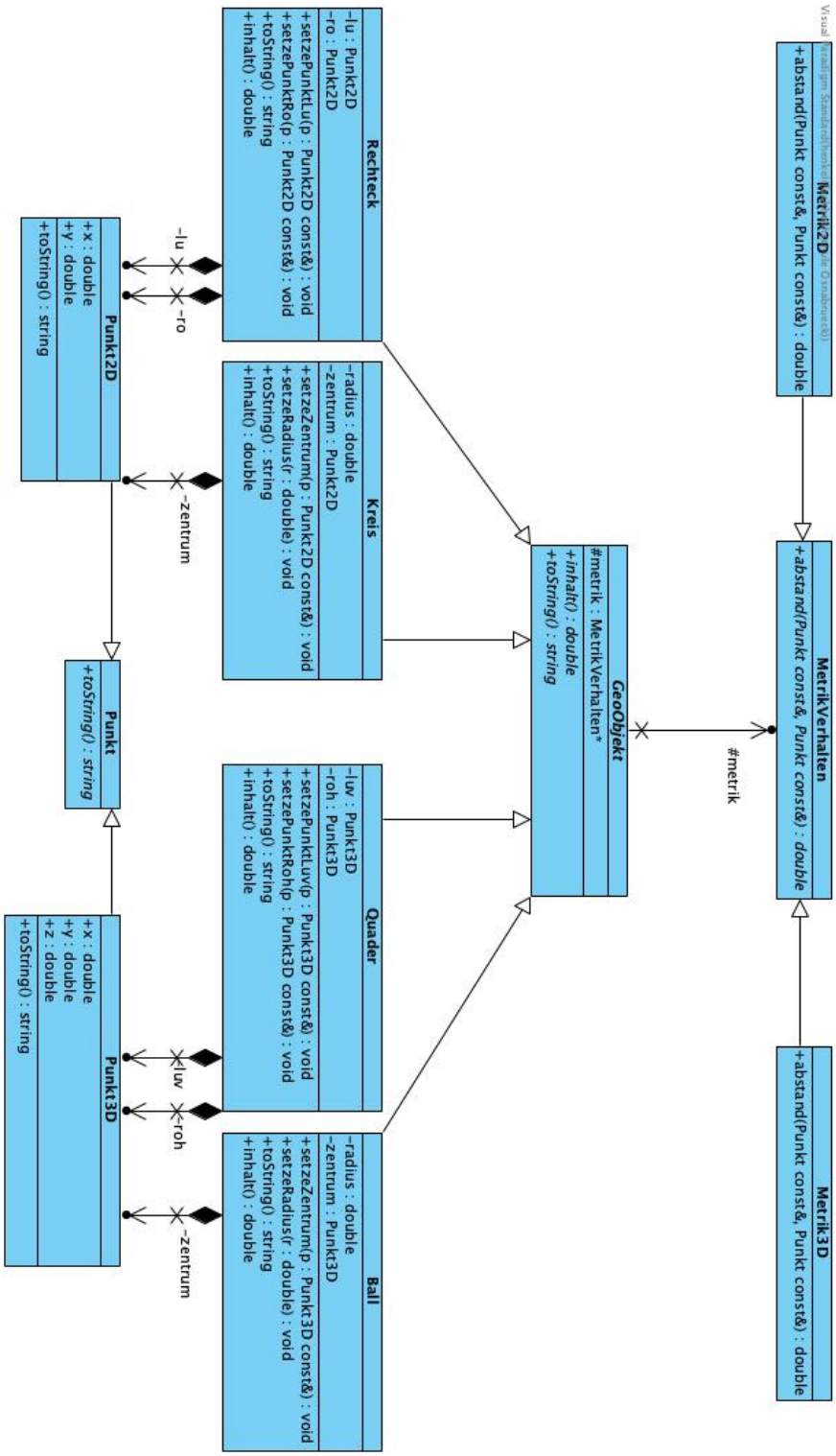
- `Punkt` ist ein Interface zur Bereitstellung von Punkten in der Ebene oder im Raum
- `MetrikVerhalten` ist ein Interface, dessen Implementationen die Berechnung des euklidischen Abstands zweier Punkte durch die Methode `abstand` bereitstellt
- `GeoObjekt` ist eine abstrakte Basisklasse. Das geschützte Element `metrik` ermöglicht in den konkreten Ableitungen passende Abstandsbestimmungen. Die öffentliche rein virtuelle Methode `inhalt` soll in den Ableitungen den Flächeninhalt bzw. das Volumen berechnen.
- Zu diesem Zweck besitzen die konkreten Ableitungen `Rechteck`, `Kreis`, `Quader`, `Ball` entsprechende Daten und Methoden

Die konkreten GeoObjekte sollen polymorph kopiert- und zugewiesen werden können. Die zugehörigen Standardmethoden müssen also noch zusätzlich implementiert werden. Inkompatible Zuweisungen sollen durch Werfen eines Ausnahmeobjektes vom Typ `OperandenPassenNicht` (leere Ausnahmeklasse) abgefangen werden.

Das Testprogramm `main.cpp` soll die folgende Ausgabe produzieren (die Zahl hinter den eckigen Klammern ist jeweils der berechnete Inhalt).

```
tatischer Test
Rechteck: [(0, 0), (2, 1)] 2

Polymorphietest - tiefe Kopie mittels clone
Rechteck: [(0, 0), (2, 1)] 2
Rechteck: [(-1, -1), (2, 1)] 6

Polymorphietest - tiefe Kopie mittels assign
Rechteck: [(0, 0), (2, 1)] 2
Rechteck: [(0, 0), (2, 2)] 4

Kreise
---------
statischer Test
```

```
Kreis: [(1, 1), 2] 12.5664

Polymorphietest - tiefe Kopie mittels clone
Kreis: [(1, 1), 2] 12.5664
Kreis: [(1, 1), 1] 3.14159

Polymorphietest - tiefe Kopie mittels assign
Kreis: [(1, 1), 2] 12.5664
Kreis: [(0, 0), 2] 12.5664
erwartete Ausnahme geworfen
erwartete Ausnahme geworfen

Quader
---------
statischer Test
Quader: [(0, 0, 0), (2, 1, 2)] 4

Polymorphietest - tiefe Kopie mittels clone
Quader: [(0, 0, 0), (2, 1, 2)] 4
Quader: [(-1, -1, -1), (2, 1, 2)] 18

Polymorphietest - tiefe Kopie mittels assign
Quader: [(0, 0, 0), (2, 1, 2)] 4
Quader: [(0, 0, 0), (2, 2, 2)] 8
erwartete Ausnahme geworfen

Baelle
---------
statischer Test
Ball: [(1, 1, 1), 2] 33.5103

Polymorphietest - tiefe Kopie mittels clone
Ball: [(1, 1, 1), 2] 33.5103
Ball: [(1, 1, 1), 1] 4.18879

Polymorphietest - tiefe Kopie mittels assign
Ball: [(1, 1, 1), 2] 33.5103
Ball: [(0, 0, 0), 2] 33.5103
erwartete Ausnahme geworfen
```


Rechteck
-lu : Punkt2D
-ro : Punkt2D
+setzePunktLu(p : Punkt2D const&) : void
+setzePunktRo(p : Punkt2D const&) : void
+toString() : string
+inhalt() : double
Kreis
-radius : double
-zentrum : Punkt2D
+setzeZentrum(p : Punkt2D const&) : void
+setzeRadius(r : double) : void
+toString() : string
+inhalt() : double
Quader
-luv : Punkt3D
-roh : Punkt3D
+setzePunktLuv(p : Punkt3D const&) : void
+setzePunktRoh(p : Punkt3D const&) : void
+toString() : string
+inhalt() : double
Ball
-radius : double
-zentrum : Punkt3D
+setzeZentrum(p : Punkt3D const&) : void
+setzeRadius(r : double) : void
+toString() : string
+inhalt() : double
Punkt2D
+x : double
+y : double
+toString() : string
Punkt
+toString() : string
Punkt3D
+x : double
+y : double
+z : double
+toString() : string
GeoObjekt
#metrik : MetrikVerhalten*
+inhalt() : double
+toString() : string
MetrikVerhalten
+abstand(Punkt const&, Punkt const&) : double
Metrik2D
+abstand(Punkt const&, Punkt const&) : double
Metrik3D
+abstand(Punkt const&, Punkt const&) : double
-lu
-ro
-zentrum
-luv
-roh
-zentrum
#metrik