Einführung in die Objektorientierte Programmierung: Klassen, Methoden, Attribute, Getter, Setter, Schnittstellen

AUFGABE 1.1

Ziel dieser Übung ist es, das Beispiel aus der Vorlesung zu wiederholen, dass die Begriffe Objekt und Klasse veranschaulicht.

Ziel ist es, eine Klasse *Rechteck* zu definieren, die eine (Computer-)Abstraktion eines Rechtecks darstellt mittels: Länge, Breite und Flächeninhalt.

(a) Erstellen Sie zunächst eine Datei *Rechteck.cpp* und definieren Sie darin die Klasse.
(b) Verwenden Sie Ihre allererste Klasse, indem Sie eine Instanz in *main()* erstellen.
(c) Implementieren Sie nun die "Rechteck"-Abstraktion, indem Sie zuerst zwei Attribute Breite und Länge hinzufügen.
(d) Implementieren Sie eine Methode, welche die Oberfläche berechnet - eine klassenspezifische Funktion.
(e) Sie können jetzt versuchen, Ihre Klasse zu testen: Fügen Sie in *main()* jedem der Felder eine Zuweisung und einen Aufruf der Methode *oberfl()* hinzu.
(f) Versuchen Sie, Ihr Programm zu kompilieren. Was ist passiert?

Hinweis: Die Botschaft ist ziemlich klar: Alle Attribute sind privat. Wenn kein Bezeichner Angegeben wird, werden die Mitglieder (Attribute/Methoden) als standardmäßig privat betrachtet. Das bedeutet, dass sie nicht außerhalb des Objekts verwendet werden können.

Insbesondere dürfen Sie *recht.breite* nicht verwenden! Es ist nicht außerhalb der Klasse definiert. Um ein Attribut außerhalb der Klasse zugänglich zu machen, muss es als *public* deklariert werden.

(g) Versuchen Sie nun Attribute privat und nur Methoden öffentlich machen. Was passiert und aus welchem Grund?
(h) Richten Sie nun „get“ und „set“ Methoden ein.
(i) Schließlich können Sie die Methoden, die die Attribute des Objekts nicht ändern (man spricht von „Prädikat“-Methoden), explizit markieren, indem Sie das Wort *const* hinter der Deklaration der Argumente entsprechender Methoden hinzufügen.

AUFGABE 1.2 (Kreise)

Das Ziel dieser Übung ist es, eine Klasse zu schreiben, die den Begriff des Kreises darstellt.

Schreiben Sie ein Programm *Kreis.cpp*, in dem Sie eine Klasse *Kreis* definieren, die als Attribute einen Radius und die Koordinaten des Mittelpunkts (des Kreises) hat.

(a) Deklarieren Sie dann die entsprechenden „get“- und „set“-Methoden, zum Beispiel:
```
void getMittelpunkt(double& x, double& y) const { ... }
void setMittelpunkt(double x, double y) { ... }
```
Hinweis: Dies ist nur ein Beispiel unter vielen. Wenn Sie andere Prototypen für diese Methoden bevorzugen, können Sie Ihre Lösung gerne implementieren. Insbesondere können Sie Ihr Programm nach den Übungen 4 und 5 überarbeiten.

(b) Fügen Sie dann folgende Methoden als Teil der Schnittstelle hinzu:
- `double flache() const`, berechnet die Fläche des Kreises und gibt sie zurück (Pi mal das Quadrat des Radius);
- `bool istImInneren(double x, double y) const`, testet, ob der als Parameter übergebene Punkt mit den Koordinaten (x,y) Teil des Kreises ist oder nicht (inkl. Rand). Die Methode gibt wahr zurück, wenn der Test positiv ist, andernfalls falsch.

(c) Instanziieren Sie in `main()` zwei Objekte der Kreis-Klasse. Weisen Sie ihren Attributen Werte Ihrer Wahl zu und testen Sie Ihre `flache`- und `istImInneren` -Methoden.

AUFGABE 1.3 (Zaubertrick)

Hier wollen wir ein Programm schreiben, das den folgenden elementaren Zaubertrick „simuliert“:

Ein Zauberer bittet einen Zuschauer, sein Alter und den Betrag, den er in der Tasche hat (weniger als 100 Euro), auf einen Zettel zu schreiben.

Dann bittet er den Zuschauer, das Papier seinem Assistenten zu zeigen, der es lesen muss (ohne etwas zu sagen), und im Stillen folgende Rechnung durchführen soll: das Alter mit 2 multiplizieren, 5 dazuzählen, das Ergebnis mit 50 multiplizieren, den Betrag aus der Tasche dazu addieren und dann die Anzahl der Tage eines Jahres abziehen. Der Assistent spricht dann das Ergebnis laut aus.

Indem der Zauberer gedanklich (schnell!) 115 zu der erhaltenen Zahl hinzufügt, findet der Zauberer sofort das Alter und den Betrag in seiner (geheim gebliebenen) Tasche.

(a) Modellieren Sie diesen Zaubertrick, indem Sie mindestens die (einfachen) Klassen Magier, Zauberer und Zuschauer definieren. Es könnte auch nützlich sein, eine Klasse Papier zu haben. Die Zuschauer-Instanz sollte den Programmbenutzer nach seinem Alter und dem Geldbetrag in seiner Tasche fragen und sicherstellen, dass der richtige Wert eingegeben wird (zwischen 0 und 99). Versuchen Sie, eine möglichst genaue Modellierung zu ermöglichen (insbesondere von Zugriffsrechten Gebrauch machen, wo dies relevant erscheint).

(b) Zeigen Sie für jede Methode auf dem Bildschirm die laufende Operation und den Akteur an, der sie ausführt.
Hinweis: Es gibt viele Variationsmöglichkeiten. Beginnen Sie mit einem sehr einfachen Modell und entwickeln Sie es weiter, um der beschriebenen „realen“ Situation näher zu kommen.

Ausgabebeispiel:

```
[Zuschauer] (Ich betrete die Szene)
Wie alt bin ich? 35
Wie viel Geld habe ich in meiner Tasche (<100)? 112
Wie viel Geld habe ich in meiner Tasche (<100)? 12
[Zuschauer] (Ich bin hier)
[Magier] ein kleiner Zaubertrick...
[Zuschauer] (Ich schreibe das Papier)
[Zuschauer] (Ich zeige das Papier)
[Assistent] (Ich lese das Papier)
[Assistent] (Ich rechne im Kopf)
[Assistent] Ich sage: 3397!
[Zauberer]
- hmm... Ich sehe, dass du 35 Jahre alt bist und 12 Euro in
der Tsche hast!
```

Hinweis:
Die „Objekte“ des Programms wurden bereits vorgeschlagen: Magier, Assistent, Zuschauer und möglicherweise Papier; Denken Sie dann über die Attribute nach: „Wer hat was? » und bei den Methoden: « Wer macht was? ». Versuchen Sie, eine Methode für jede durchgeführte Elementaraktion zu definieren: erste Aktionen des Zuschauers (nach Alter und Geldbetrag fragen), auf das Papier schreiben, das Papier zeigen; Aktionen des Assistenten: das Papier lesen (das er daher erhalten muss), die Berechnung durchführen; usw.

AUFGABE 1.4 (3D-Punkte)

Das Ziel dieser Übung ist es, auf elementare Weise eine Klasse zu implementieren, die die Koordinaten im Raum (3D) darstellt.

(a) Definieren Sie in der Datei `Punkt3D.cpp` die Klasse `Punkt3D`, die einen Punkt im Raum durch seine drei Koordinaten darstellt und die folgenden Methoden hat:
- „`init`“, wird verwendet, um die drei Koordinaten eines Punkt3D-Objekts aus drei als Parameter empfangenen Werten vom Typ Double zu initialisieren;
- „`Anzeige`“, ermöglicht die Anzeige von Koordinaten eines Punkt3D-Objekts;
- und „`Vergleichen`“, was es ermöglicht, das aktuelle Objekt mit einem anderen als Parameter übergebenen Objekt vom Typ Punkt3D zu vergleichen (die Gleichheit der Koordinaten zu testen).

(b) Erstellen Sie im Wesentlichen drei Punkte (drei Instanzen), darunter zwei mit identischen Koordinaten, und testen Sie die beiden vorherigen Methoden.

AUFGABE 1.5 (Dreiecke)

Schreiben Sie basierend auf dem Programm aus der vorherigen Übung ein neues Programm dreiecke.cpp, das dem Benutzer ermöglicht, die Koordinaten (x, y und z) der Eckpunkte eines Dreiecks (in 3D) einzugeben. Das Programm zeigt dann den Umfang des Dreiecks zusammen mit einer Meldung an, die angibt, ob es sich um ein gleichschenkliges Dreieck handelt oder nicht.

Hinweise:

- Ein Dreieck ist gleichschenklig, wenn mindestens zwei Seiten gleich lang sind.
- Die Formel zur Berechnung des Abstands zwischen zwei Punkten im Raum (x1, y1, z1) und (x2, y2, z2) lautet: die Quadratwurzel (sqrt-Funktion von cmath) von ((x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2).
- Der Umfang eines Dreiecks kann als Summe der Abstände zwischen den drei Eckpunkten berechnet werden.

Ausgabebeispiel:

```
Eingabe eines neuen Punktes
Bitte geben Sie x ein: 0
Bitte geben Sie y ein: 0
Bitte geben Sie z ein: 0
Eingabe eines neuen Punktes
Bitte geben Sie x ein: 2.5
Bitte geben Sie y ein: 2.5
Bitte geben Sie z ein: 0
Eingabe eines neuen Punktes
Bitte geben Sie x ein: 0
Bitte geben Sie y ein: 5
Bitte geben Sie z ein: 0
Umfang: 12.071067811865476
Dieses Dreieck ist gleichschenklig!
```

Weitere Hinweise:

- Denken Sie an die Objekte, die Sie im Programm verwenden müssen. Sie könnten das Dreieck beispielsweise durch eine Klasse Dreieck darstellen, die die Längen seiner drei Seiten enthält, da wir nicht die Ecken des Dreiecks als solche verwenden, sondern einfach die Längen seiner Seiten.
- Denken Sie auch an Methoden, die für die Klassen `Dreieck` und `Punkt3D` nützlich wären (die nur vorübergehend zum Initialisieren eines Dreiecks verwendet würden).
- Definieren Sie eine Abstandsfunktion, die zwei `Punkt3D`s als Argumente akzeptiert und deren Abstand zurückgibt.

AUFGABE 1.6 (BMI) - **Abgabe**

Das Ziel dieser Übung ist es, „Patienten“ zu erstellen, die ein Gewicht und eine Größe haben, und ihren „Body Mass Index“ (BMI) zu berechnen.

Laden Sie zum Starten den auf StudIP verfügbaren Quellcode herunter.

Der bereitgestellte Code bewirkt Folgendes:

- erstellt einen Patienten;
- liest das Gewicht und die Größe des Patienten aus der Standardeingabe;
- zeigt die Patientendaten und den BMI an.

Die letzten beiden Schritte werden wiederholt, solange weder das Gewicht noch die Höhe Null sind.

Die Implementierung der Patient-Klasse fehlt. Das müssen Sie bereitstellen. Ein Patient wird durch sein Gewicht und seine Größe charakterisiert.

Patientenspezifische Methoden sind:

- eine Methode `init`, die zwei `double` als Parameter nimmt, das erste, um das Patientengewicht und das zweite für seine Körpergröße zu initialisieren; Diese Daten werden nur übernommen wenn sie positiv sind und dem Patienten zugeordnet; wenn nicht, werden Gewicht und Höhe beide mit Null initialisiert; um zu vereinfachen, werden diese Daten nicht weiter überprüft;
- eine Methode-`anzeigen`, die auf dem Terminal die Attribute des Patienten anzeigt, mit folgendem Format: Patient: <Gewicht> kg für <Größe> m. Dabei ist <Gewicht> durch das Gewicht des Patienten und <Größe> durch dessen Größe zu ersetzen. Diese Ausgabe soll mit einem Zeilenumbruch enden.
- eine Methode `gewicht`, die das Gewicht des Patienten zurückgibt;
- eine Methode `groesse`, die die Körpergröße des Patienten zurückgibt;
- eine Methode `bmi`, die den BMI des Patienten zurückgibt: sein Gewicht dividiert durch das Quadrat seiner Höhe; Wenn die Höhe null ist, gibt die Methode null zurück.

Diese Methoden sollen Teil der öffentlichen Schnittstelle der Klasse sein.

Ausgabebeispiel*:*

```
Gewicht (kg) und Größe (m) eingeben: 80,0 1,7
Patient: 80 kg für 1,7 m
BMI: 27,6817
Gewicht (kg) und Größe (m) eingeben: 56,5 1,8
Patient: 56,5 kg für 1,8 m
BMI: 17,4383
Gewicht (kg) und Größe (m) eingeben: 0 0
Patient: 0 kg für 0 m
BMI: 0
```

AUFGABE 1.7 (Sparschwein) - **Abgabe**

Das Ziel dieser Übung ist es, ein Sparschwein zu simulieren, in das wir Geld einzahlen und von dem wir Geld abheben. Wir möchten damit bestimmte Beträge bezahlen.

Laden Sie zum Starten den auf StudIP verfügbaren Quellcode herunter.

Der bereitgestellte Code erstellt ein Sparschwein und manipuliert es (leeren, schütteln, Inhalt anzeigen usw.).

Dieses Programm fragt den Benutzer auch nach dem Budget, das er in seinen Urlaub investieren möchte.

Wenn das Sparschwein genug Geld enthält (das Budget oder mehr), gibt es an, wie viel Geld nach den Ferien übrig bleiben würde. Im umgekehrten Fall gibt es den fehlenden Betrag an, um mit dem gewünschten Budget in den Urlaub zu fahren.

Die Implementierung der Klasse `Sparschwein` fehlt und Sie müssen sie bereitstellen.

Ein Sparschwein zeichnet sich durch die Summe aus, die es enthält.

Sparschwein-spezifische Verfahren sind:

- eine Methode `getBetrag`, die den Betrag des Sparschweins zurückgibt;
- ein Methode `anzeigen`, die die Informationen über das Sparschwein in folgendem Format anzeigt:
  - « Sie sind pleite » wenn das Sparschwein leer ist
  - « Sie haben <Betrag> Euro in Ihrem Sparschwein“ im umgekehrten Fall, wobei <Betrag> der Geldbetrag im Sparschwein ist.
- eine Methode `schuetteln`, die im Terminal die Meldung „Bling bling“ gefolgt von einem Zeilenumbruch anzeigt, falls das Sparschwein nicht leer ist, und sonst nichts anzeigt;
- eine Methode `fuellen`, die eine bestimmte Menge, angegeben als `double` Parameter, in das Sparschwein hinzufügt; nur die positiven Beträge werden akzeptiert (ansonsten wird nichts unternommen);
- eine Methode `leeren`, die das Sparschwein auf Null setzt.
- Eine Methode `entnehmen`, die es erlaubt einen Betrag (als Parameter angegeben) aus dem Sparschwein zu entnehmen. Ist der Wert negativ, soll nichts passieren. Ist der Betrag größer als der Betrag in dem Sparschwein, soll das Sparschwein komplett geleert werden und diese Methode gibt nichts zurück.
- Eine Methode `ausreichend_geld`, mit zwei Parametern, die `true` zurückgibt, wenn das Sparschwein genug Geld enthält, um eine bestimmte Summe auszugeben (die als erster Parameter (vom Typ double) angegeben wurde), andernfalls `false`; Wenn genug Geld vorhanden ist, enthält der zweite Parameter `saldo` (vom Typ double und als Referenz übergeben) den Geldbetrag, der im Sparschwein verbleiben würde, wenn wir das angegebene Budget ausgeben. Andernfalls enthält `saldo` den Betrag (eine positive Zahl), der im Sparschwein fehlt, um das vorgegebene Budget zu erreichen; Wenn das Budget negativ oder null ist, gibt die Methode `ausreichend_geld` true zurück und dem zweiten Parameter `saldo` wird der Geldbetrag zugewiesen, der im Sparschwein enthalten ist: es ist genau so, als ob wir nach einem Null-Budget fragen würden.

Diese Methoden werden Teil der öffentlichen Schnittstelle des Sparschweins sein.

Ausgabebeispiel:

```
Sie sind pleite.
Sie sind pleite.
Bling Bling
Sie haben: 550 Euro in Ihrem Sparschwein.
Sie haben: 535 Euro in Ihrem Sparschwein.
Geben Sie das Budget Ihres Urlaubs an: 1000,0
Ihnen fehlen 465 Euro für den Urlaub!
```

Oder

```
Sie sind pleite.
Sie sind pleite.
Bling Bling
Sie haben: 550 Euro in Ihrem Sparschwein.
Sie haben: 535 Euro in Ihrem Sparschwein.
Geben Sie das Budget Ihres Urlaubs an: 400,0
Du bist reich genug, um in den Urlaub zu fahren!
Nach dem Urlaub bleiben Ihnen 135 Euro übrig.
```