

Programmieren 1 Übung: Geometrische Objekte: Vererbung: Rechtecke und Ellipsen

Klaus Kusche

In Fortsetzung des vorigen Beispiels

(nimm die Musterlösung von Übung 16, wenn du keine eigene Ausgangsbasis hast):

Neben Rechtecken soll unser Programm auch Ellipsen zeichnen können.

Es gibt dazu die Funktion **sdldrawCirc** in meinem **sdlinterf**.

Die Ellipsen sollen eine eigene Klasse **Circ** werden (wieder in separaten Files).

Da alle Member-Variablen und ein Großteil des Codes von **Circ** ident zu **Rect** sind (nur **draw** und **undraw** müssen statt **sdldrawRect** jetzt **sdldrawCirc** aufrufen), und da wir Rechtecke und Ellipsen in unserem Hauptprogramm gemeinsam behandeln können wollen (sie haben ja beide genau dieselbe Schnittstelle, d.h. dieselben Methoden), leiten wir sowohl **Rect** als auch **Circ** von einer gemeinsamen Vaterklasse **GraObj** ab:

- **GraObj** enthält alles, was alle abgeleiteten Klassen gemeinsam haben. In unserem Fall sind das alle Member-Variablen und alle Methoden (auch **draw** und **undraw** haben zwar verschiedene Implementierungen, aber in allen Klassen den gleichen Prototypen). Es ist daher am einfachsten, wenn du die Klasse **Rect** aus der alten Musterlösung in **GraObj** umbenennst.
- **Rect** und **Circ** enthalten nur jene Dinge, in denen sich Rechtecke und Ellipsen unterscheiden. Das sind
 - die Implementierungen von **draw** und **undraw**,
 - sowie der jeweilige Konstruktor und Destruktor.

Details dazu in den Hinweisen!

Beide Klassen sind bis auf den Namen und die SDL-Aufrufe ident. Am schnellsten geht es, wenn du eine davon schreibst und dann kopierst und änderst.

Hinweise:

- Auf die Member-Variablen für Farbe, Position und Größe sollen auch die abgeleiteten Klassen direkt zugreifen können, nicht aber "fremder" Code.
Alle Member für die Verwaltung der Z-Reihenfolge dürfen hingegen nur für GraObj sichtbar sein: Es soll abgeleiteten Klassen nicht möglich sein, direkt in den Z-Mechanismus einzugreifen.
- Überlege dir, wie die Konstruktoren von **Rect** und **Circ** aussehen müssen:
 - Alle Membervariablen sind schon in der Basisklasse deklariert und werden daher auch wie bisher in der Initialisierungsliste des Konstruktors der Basisklasse GraObj initialisiert (man kann in einer Initialisierungsliste nur Member der eigenen Klasse initialisieren, nicht geerbt!).

- Die Initialisierungsliste in den Konstruktoren der abgeleiteten Klassen initialisiert keine Member, sondern ruft nur die Initialisierung der Basisklasse auf.

Was musst du dafür in die Initialisierungsliste schreiben?!

- Die Aufrufe von **draw** und **undraw** passieren im Code des Konstruktors und Destruktors der abgeleiteten Klasse (weil sie ja das **draw** und **undraw** der jeweiligen abgeleiteten Klasse aufrufen sollen, nicht das der Basisklasse), in der Basisklasse enthalten Konstruktor und Destruktor vorläufig keinen Code (auch kein **draw** und **undraw**).
- Dasselbe gilt für den Destruktor, aus demselben Grund: Ein Teil des Codes kann in **GraObj** bleiben (welcher?), ein Teil gehört in die abgeleiteten Klassen.

Wie muss der Destruktor deklariert werden, damit auch der Code-Teil in den abgeleiteten Klassen ausgeführt wird, wenn man ein Objekt löscht, von dem man nur weiß, dass es von **GraObj** abgeleitet ist?

- Derzeit brauchen wir nur 2 virtuelle Methoden. Welche und warum?

In späteren Übungen werden wir auch noch **setPos**, **setSize**, **move**, **scale** und **rotate** überschreiben.

- **GraObj** selbst deklariert zwar **draw** und **undraw** (denn alle davon abgeleiteten Objekte kann man zeichnen), kann aber selbst keinen Code dafür enthalten (weil die Klasse **GraObj** nicht weiß, wie ein **Rect**, **Circ** oder zukünftige weitere Klassen gezeichnet werden).

GraObj ist daher eine abstrakte Klasse, es hat keinen Sinn, ein **GraObj**-Objekt direkt anzulegen, das zu keiner der abgeleiteten Klasse gehört.

Bring das in deinem Code entsprechend zum Ausdruck! (wie?)

- Schreib wieder einen separaten **.cpp**- und **.h**-File pro Klasse und achte darauf, überall die richtigen Files zu inkludieren (und nicht zu viele).

Schreib dazu wieder ein Hauptprogramm zum Testen

(du kannst als Ausgangsbasis auch mein **main** aus der vorigen Übung verwenden).

Ändere zuerst einmal einfach **Rect** auf **Circ**. Klappt das?

Damit man die Vererbung und den **virtual**-Mechanismus auch “bei der Arbeit” sieht, sollte dein Hauptprogramm dann ein Array von Pointern auf **GraObj**-Objekte enthalten, von denen einige auf **Rect**-Objekte und einige auf **Circ**-Objekte zeigen (d.h. jeder Pointer im Array wird z.B. zufällig entweder mit einem **new Rect(...)** oder einem **new Circ(...)** initialisiert).

Wende dann auf alle Objekte im Array irgendwelche grafischen Operationen an.

Nur zum Testen (damit man sieht, was ohne **virtual** passiert):

Implementiere **draw** und **undraw** in **GraObj** selbst so, dass nur der Mittelpunkt des Objektes gezeichnet wird, und lass dann probierhalber das **virtual** weg!

Was passiert, wenn zwar **draw** und **undraw** **virtual** sind, aber der Destruktor nicht, und wenn du dann deine Objekte im **GraObj**-Array mit **delete** wieder freigibst?