

Programmieren C++: Vererbung, `clone`: Grafik mit Ellipsen

Klaus Kusche

Dieses Beispiel ist eine Fortsetzung des vorigen Beispiels, nimm deine Lösung (oder meine Musterlösung) des vorigen Schrittes als Ausgangsbasis, und zwar in der Version ohne (vor) dem speziellen Copy-Konstruktor und `moveOnTop!` (d.h. die Variante mit dem Array Ping-Pong spielender, dynamisch angelegter Rechtecke).

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`: Sie 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!

Alles andere erben `Rect` und `Circ` unverändert von ihrer Vaterklasse `GraObj`.

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, Größe und Fluggeschwindigkeit sollen auch die abgeleiteten Klassen direkt zugreifen können, nicht aber "fremder" Code.
- Überlege dir, wie die Konuktoren von `Rect` und `Circ` aussehen müssen:
 - Alle Membervariablen sind schon in der Basisklasse deklariert und werden daher auch wie bisher in der Initialisierungsliste des Konuktors der Basisklasse GraObj initialisiert (man kann in einer Initialisierungsliste nur Member der eigenen Klasse initialisieren, nicht geerbte!).
 - Die Initialisierungsliste in den Konuktoren der abgeleiteten Klassen initialisiert keine Member, sondern ruft nur die Initialisierung der Basisklasse auf.

Was gehört dafür als erster und einziger Eintrag in die Initialisierungsliste?

- 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**).
- Wir wollen **Rect** und **Circ** auch einen ganz normalen Copy-Konstruktor geben: Er soll eine identische, unveränderte Kopie des Original-Objektes erzeugen und diese Kopie auch gleich anzeigen.
 - Auch hier gilt: Für das Anzeigen ist die abgeleitete Klasse zuständig, für die Initialisierung der Member die Basisklasse.
 - In den abgeleiteten Klassen müssen wir daher zum Zeichnen auf jeden Fall einen expliziten Copy-Konstruktor implementieren.

Überlege: Braucht die Basisklasse zum identen Kopieren der Member auch einen explizit programmierten Copy-Konstruktor?
- Dasselbe gilt für den Destruktor, aus demselben Grund: Das **undraw** gehört in die abgeleiteten Klassen.
Wie muss der Destruktor in **GraObj** deklariert werden, damit der Destruktor 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**, **fly** und **scale** 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, und daher selbst nichts zeichnen kann).
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).

Beginne Deine Tests mit dem Hauptprogramm aus der alten Übung:

- Es müsste eigentlich unverändert funktionieren?! Ebenso müsste es funktionieren, wenn man **Rect** überall durch **Circ** ersetzt (und natürlich **circ.h** inkludiert).
- Damit man die Vererbung und den **virtual**-Mechanismus “bei der Arbeit” sieht, sollte dein Hauptprogramm **Rect**’s und **Circ**’s gleichzeitig fliegen lassen, aber trotzdem nur ein einziges Array für beide Arten von Objekten verwenden.

Ersetze dazu den Aufruf von **new Rect** in der Schleife zum Befüllen des Arrays durch eine Hilfsfunktion (ohne Parameter). Sie soll intern zufällig und mit gleicher Wahrscheinlichkeit entweder ein **Rect**- oder ein **Circ**-Objekt dynamisch anlegen und einen Pointer auf das neue Objekt als Returnwert liefern.

Überlege: Mit welchem Typ musst du den *Retruntyp dieser Hilfsfunktion* und das *Array im Hauptprogramm* deklarieren, wenn beide sowohl auf **Rect**- als auch auf **Circ**-Objekte zeigen können?

Obwohl die “Herumflieg-Schleife” im Hauptprogramm jetzt nicht mehr weiß, ob sie es mit einem **Rect** oder einem **Circ** zu tun hat (weil man das aus dem Typ des Arrays nicht mehr ablesen kann), müsste immer noch alles wie bisher funktionieren.

- *Nur zum Testen*, wenn Zeit ist (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 probehalber das **virtual weg!** Was passiert?

Was passiert, wenn zwar **draw** und **undraw virtual** sind, aber der *Destruktor nicht*, und wenn du deine Objekte im **GraObj**-Array mit **delete** wieder *freigibst*? (Nachher beide Änderungen wieder rückgängig machen!)

clone

Eines ist aber *noch nicht möglich*: Das Array zu befüllen, indem man das *erste Objekt zufällig* als **Rect** oder **Circ** erzeugt und *alle weiteren Objekte als Kopie* des jeweils vorigen Objektes: Es gibt keinen Copy-Konstruktor, der von beiden Arten des Original-Objektes eine Kopie erzeugen kann, man muss für **Rect**-Objekte den **Rect**-Copy-Konstruktor und für **Circ**-Objekte den **Circ**-Copy-Konstruktor aufrufen.

Deklariere daher in der Basisklasse eine *rein virtuelle Methode* **clone** mit dem üblichen Prototyp und der üblichen Bedeutung und implementiere sie in allen abgeleiteten Klassen *in der üblichen Art und Weise*.

Ich habe dann folgendes **Hauptprogramm** gebastelt:

- Parameter in meiner “*erzeuge zufällig ein Rect- oder Circ-Objekt*”-Hilfsfunktion: Zufällige Farbe, Position in Fenster-Mitte, Größe 10*10 bei Rechtecken und 13*13 bei Ellipsen, Fluggeschwindigkeit zufällig -7 ... +7 (wieder ohne 0).
- Ich habe (in einem **anzahl*laenge** großen Objekt-Array) **anzahl** viele “Schlangen” der Länge **laenge** erzeugt:
 - Wenn der Array-Index glatt durch **laenge teilbar** ist, habe ich mit der Hilfsfunktion ein neues, *zufälliges Objekt* erzeugt.
 - *Sonst* habe ich mit **clone** eine *Kopie des vorigen Objektes* erzeugt und dieses Objekt gleich danach mit **move** in x- und y-Richtung *um das Dreifache seiner eigenen Geschwindigkeit verschoben* (dazu muss man mit get-Methoden seine Geschwindigkeit auslesen).
- Die **Hauptschleifen** (eine Endlos-Schleife und darin eine Schleife über alle Objekte) lassen einfach *alle Objekte endlos und unverändert fliegen*.