Programmieren C++: Grafische Klassen: Abstrakte Klassen, statische Member und Methoden

Klaus Kusche

Diese Übung baut auf dem Beispiel aus der vorigen Übung auf.
Nimm die *Musterlösung der vorigen Übung*, wenn du keine eigene Ausgangsbasis hast!

1. Schritt: Rein virtuelle Methoden, abstrakte Klassen

In der vorigen Übung haben wir besprochen, dass

- eine Implementierung der Methoden **`draw`** und **`undraw`** in der Klasse **`GraObj`** eigentlich nicht sinnvoll ist, weil wir nicht wissen, was wir zeichnen müssen (ein Rechteck, eine Ellipse oder sonstwas),
- und es aus diesem Grund auch *gar nicht möglich sein sollte*, Objekte der Klasse **`GraObj`** direkt zu erzeugen, sondern *nur* Objekte davon *abgeleiteter Klassen*.

Damals kannten wir die dazu notwendigen C++-Konstrukte noch nicht. Wir haben uns damit beholfen, dass ein **`GraObj`** bei **`draw`** einfach nur seinen Mittelpunkt zeichnet (damit wir sehen, wenn das **`draw`** von **`GraObj`** versehentlich doch aufgerufen wird).

Inzwischen habt ihr die notwendigen Dinge gelernt:
Mach aus **`draw`** und **`undraw`** *rein virtuelle Methoden*. Prüfe auch die Auswirkung:

- Funktioniert dein Programm mit Rechtecken und Ellipsen noch genauso?
- Kannst du eine Variable der Klasse **`GraObj`** definieren?
- Kannst du eine Variable deklarieren, die ein Pointer auf **`GraObj`** ist?
- Kannst du ein **`new GraObj(...)`** machen?

Überlege vor dem Probieren die Antwort!

2. Schritt: Statische Member und Methoden

Das Zeichnen unserer grafischen Objekte hat 2 Schwächen:

- Wenn ein Objekt weggelöscht oder bewegt wird, hinterlässt es *"Löcher"* in darunterliegenden Objekten.
- Wir können nicht kontrollieren, in welcher *Reihenfolge* übereinander liegende Objekte gezeichnet werden:
  Das zuletzt angelegte oder bewegte Objekt liegt immer obenauf.

Das wollen wir lösen, und zwar wie folgt:

- Man kann *zu jedem grafischen Objekt einen Tiefenwert* (*"Z-Wert"*, ein **`int`**) angeben.

  Dieser legt fest, wie weit vorne oder hinten das Objekt im Gesamtbild liegt, d.h. welche anderen Objekte es verdeckt und von welchen es verdeckt wird.

  Für diesen Wert gibt es *eine Set- und eine Get-Methode*: **`setZ`** und **`getZ`**.

- Alle Objekte mit Z-Wert werden *geordnet* (sortiert) in ein *Array* eingetragen.
- Eine Methode **`drawAll`** geht dieses Array durch (von den im Bild ganz hinten liegenden niedrigen Z-Werten zu den vordersten hohen) und *zeichnet alle darin enthaltenen Objekte* in dieser Reihenfolge neu.
- Wurde zu einem neuen Objekt noch kein Z-Wert angegeben, oder wird der Z-Wert mit **`setZ`** auf **`0`** gesetzt, so ist das Objekt nicht im Z-Array enthalten (bzw. wird aus dem Array entfernt) und wird daher auch von **`drawAll`** nicht frisch gezeichnet.

Überlege:

1.) Der Z-Mechanismus soll für *alle* grafischen Objekte gleich funktionieren.

In welcher Klasse gehört er daher implementiert?

2.) Wir haben *3 neue Member-Variablen*: Den Z-Wert des jeweiligen Objektes, das nach Z-Werten sortierte Array von Objekten, und die Anzahl der im Array gerade enthaltenen Objekte.

Welche dieser Variablen sind pro Objekt, welche sind für die ganze Klasse gemeinsam? Wie musst du sie daher deklarieren?

Weiters haben wir 3 neue Methoden: **`getZ`**, **`setZ`** und **`drawAll`**.

Welche dieser Methoden werden für ein ganz bestimmtes Objekt aufgerufen, welche für die ganze Klasse (ohne ein bestimmtes Objekt zu haben)?

3.) *Welchen Typ* werden die Array-Elemente haben, wenn sie auf *beliebige* grafische Objekte verweisen sollen?

Im Detail sind dazu folgende Schritte notwendig:

- Zuerst einmal braucht man eine *Konstante für die Array-Größe* (diese darf fix festgelegt werden), am besten in der Klasse selbst.
- Weiters braucht man *eine Membervariable für den Z-Wert* eines jeden Objektes und *zwei Membervariablen für das Array und die Anzahl* der darin gerade enthaltenen Objekte (siehe 2. und 3. oben!).

  Alle Member für die Verwaltung der Z-Reihenfolge dürfen *nur* für **`GraObj`** sichtbar sein: Es soll abgeleiteten Klassen nicht möglich sein, direkt in den Z-Mechanismus einzugreifen.

  Erinnere dich: Für Klassenvariablen braucht man nicht nur eine *Deklaration* im **`class`**, sondern auch eine *Definition* außerhalb von **`class`**.
  Auch die *Initialisierung* muss in der Definition gemacht werden (die Anzahl der Array-Elemente ist am Anfang **`0`**!).
- Dann braucht man Deklarationen für die *Methoden* **`getZ`** (Code gleich inline!) und **`setZ`** sowie für die Klassen-Methode **`drawAll`** (Details zum Code dazu unten).
- In der *Initialisierung* neuer Objekte muss deren Z-Wert auf **`0`** gesetzt werden (wo passiert das?).
- Und wenn ein Objekt *abgebaut* wird, muss es *aus dem Z-Array entfernt* werden (Code nicht nochmals schreiben, sondern die Methode **`setZ`** mit **`0`** aufrufen!).

- **`setZ`** ist eine relativ lange Methode:
    - Wenn das Objekt schon im Array enthalten ist (schon bisher einen Z-Wert ungleich **`0`** hatte), muss es zuerst im Array gesucht und daraus *entfernt* werden.

      Dabei wird das Array um ein Element *kürzer*:
      Alle Elemente hinter dem entfernten Objekt rutschen *einen Platz nach vor*.
    - Wenn der neue Z-Wert nicht **`0`** ist, muss an der richtigen Stelle im Array ein Pointer auf das Objekt *eingefügt* werden. Dabei wird das Array größer (auf die *Maximal-Größe* achten!), und alle Elemente hinter dem neuen Objekt rutschen einen Platz *nach hinten*.

      In welche Richtung durchwandert man das Array dafür sinnvollerweise?
    - Für beides wirst du einen *Pointer auf das eigene Objekt* brauchen.

      Erinnere dich: Wie bekommt man einen Pointer auf das eigene Objekt?
    - Und schließlich muss auch die *Z-Membervariable* des Objektes selbst gesetzt werden.
- **`drawAll`** ist simpel:
    - Einmal das Array durchwandern und für jedes Objekt **`draw`** aufrufen.
- Schließlich brauchen wir wieder ein **`main`**, bei dem man den Effekt auch sieht.
    - Zuerst einmal sollte das **`main`** mehrere übereinanderliegende Objekte so zeichnen (und verändern), dass die Probleme der bisherigen Lösung sichtbar werden (d.h. entweder so viele oder so große Objekte anlegen, dass Überlappungen auftreten).
    - Dann sollten die richtigen Aufrufe von **`setZ`** und **`drawAll`** ergänzt werden. (Wie ruft man eine Klassenmethode auf?)
    - Statt die Objekte zu bewegen und die Z-Werte gleich zu lassen, kannst du wahlweise auch die Objekte ruhig liegen lassen und die Z-Werte ändern.