Programmieren C++: Objekte, die Pointer enthalten

Klaus Kusche

Wir erweitern unser Inventar-Beispiel aus der vorigen Übung:

Da es in einer Firma oft viele idente EDV-Arbeitsplätze gibt, die *aus mehreren Gegenständen* bestehen (PC, Bildschirm, Drucker, ...), wollen wir zur einfacheren Erfassung eine weitere von **`Inventar`** abgeleitete Klasse **`Sammelobj`** definieren,
die mehrere andere **`Inventar`**-Gegenstände zu einem Inventareintrag zusammenfasst.

Die Klasse **`Sammelobj`**

- ... enthält zusätzlich zu den geerbten Daten ein *Array von Pointern auf andere* **`Inventar`**-*Objekte* (das Array darf *fixe Größe* haben und *direkt als Member* im Objekt enthalten sein; **`new`** kommt erst als Zusatzaufgabe) und einen *Zähler*, der angibt, wie viele Pointer aktuell in diesem Array gespeichert sind.
- ... initialisiert dieses Array auf *leer*, wenn ein **`Sammelobj`**-Objekt neu *angelegt* wird.
- ... erzeugt *Kopien der im Array enthaltenen Gegenstände*, wenn ein **`Sammelobj`**-Objekt *kopiert* wird.
- ... *löscht die enthaltenen Gegenstände*, wenn ein **`Sammelobj`**-Objekt *gelöscht* wird.
- ... bietet eine *zusätzliche Methode* **`addItem`** zum *Hinzufügen* eines weiteren Gegenstandes zum **`Sammelobj`**-Objekt (Achtung auf die Array-Größe!).
- ... *listet* bei der virtuellen **`print`**-Methode zusätzlich *die enthaltenen Gegenstände*.
- ... liefert bei der virtuellen **`getValue`**-Methode die *Summe der Werte der enthaltenen Gegenstände* (und enthält daher selbst *keine* Wertangabe).
- ... und gibt bei der virtuellen **`setValue`**-Methode eine *Fehlermeldung* aus.

Das *Hauptprogramm* wird folgende *neue Menüpunkte* brauchen:

- *Anlegen* eines leeren **`Sammelobj`** .
- *Hinzufügen* eines bereits *bestehenden Gegenstandes* zu einem **`Sammelobj`** .

  Hier haben wir wieder das Problem, dass die Gegenstände unserer Inventarliste in einem Array von **`Inventar`**-Pointern gespeichert sind und wir bisher nicht feststellen können, *ob ein solcher Pointer nun auf ein Zubehör, einen Computer oder ein* **`Sammelobj`** *zeigt*.

  Um nicht wieder eine neue virtuelle Funktion analog zu **`isComp`** einführen zu müssen, sollten wir uns mit dem *offiziellen Konstrukt für solche Fälle* anfreunden:

  ```
  Sammelobj *soPtr = dynamic_cast<Sammelobj *> (invListe[i]);
  ```

  Zeigt **`invListe[i]`** auf ein **`Sammelobj`**, liefert **`dynamic_cast`** einen Pointer darauf. Zeigt **`invListe[i]`** auf ein Objekt einer anderen Klasse, liefert es **`NULL`**.

  Wir *entfernen* den hinzugefügten Gegenstand gleichzeitig *aus der Inventarliste des Hauptprogramms* (das ist zwar eigentlich nicht Sinn der Sache und entzieht ihn unserem direkten Zugriff, aber wenn wir ihn in der Liste des Hauptprogramms lassen würden, gerät unsere Programmlogik noch mehr durcheinander).

Zusatzaufgabe:

Wenn das klappt, bauen wir das Array-Member auf ein *dynamisch angelegtes Array* (mit **new**) ebenfalls fixer Größe um.

Da wir noch keinen Zuweisungsoperator haben, sind das *nur 4 Zeilen Änderung*: Member-Deklaration, 2 Konstruktor-Initialisierungslisten, Destruktor.

Überlege: Wenn das dynamisch angelegte Array **Inventar**-Pointer enthält, welchen Typ hat dann unser *Member*, das auf dieses dynamisch angelegte Array *zeigt*?