

Übung 2

Zeichnen mit Klassen, Lösung

Konstruktor mit Parametern für die Klasse KreisText

Der Klasse KreisText soll als erstes also ein Konstruktor mit Parametern hinzugefügt werden. Der Grund dafür ist, dass ein KreisText-Objekt genau dann erscheint, wenn ein das Objekt erzeugt wird. Um ein Objekt erscheinen zu lassen, brauchen wir ein ZeichenFlaeche-Objekt bei dem wir die „hinzufuegen“-Methoden aufrufen können.

Hier also der neue Konstruktor, mit jeweils dem Code aus der Headerdatei (.h) und der Quellcodedatei (.cpp).

```
class KreisText
{
    public:
        // Zwei Konstruktoren mit Parametern
        // Der erste nimmt zwei longs als Koordinaten
        KreisText(long x, long y, long kreisRadius,
                 ZeichenFlaeche& zeichenFlaeche);
        // Dieser zweite Konstruktor nimmt als Parameter
        // eine Koordinaten-Struktur
        KreisText(Koordinaten koordinaten, long kreisRadius,
                 ZeichenFlaeche& zeichenFlaeche);
}
```

Der Rest der Klassendefinition bleibt gleich wie bei der Lösung der letzten Übung. Hier der Code aus der .cpp Datei:

```
// Der Konstruktor muss eine Initialisierungsliste
// verwenden, denn die Elemente haben auch Konstruktoren
// mit Parametern
KreisText::KreisText(long x, long y, long kreisRadius,
                     ZeichenFlaeche& zeichenFlaeche)
    :m_Text(x,y), m_Kreis(x, y, kreisRadius)
{
    // Dei Konstante Weiss ist oben definiert!
    m_Text.setzeFarbe(Weiss);
    // Die Konstante InitText wurde oben definiert!
    m_Text.setzeText(InitText);
    // Mit diesem Konstruktor können wir die Elemente
    // direkt darstellen
    zeichenFlaeche.kreisHinzufuegen(m_Kreis);
    zeichenFlaeche.textHinzufuegen(m_Text);
}
```

```
KreisText::KreisText(Koordinaten koordinaten, long kreisRadius,
                    ZeichenFlaeche& zeichenFlaeche)
    :m_Text(koordinaten.x, koordinaten.y),
      m_Kreis(koordinaten.x, koordinaten.y, kreisRadius)
{
    // Dei Konstante Weiss ist oben definiert!
    m_Text.setzeFarbe(Weiss);
    // Die Konstante InitText wurde oben definiert !
    m_Text.setzeText(InitText);
    // Mit diesem Konstruktor können wir die Elemente
    // direkt darstellen
    zeichenFlaeche.kreisHinzufuegen(m_Kreis);
    zeichenFlaeche.textHinzufuegen(m_Text);
}
```

Der Unterschied zur ersten Lösung ist, dass wir jetzt schon im Konstruktor die Methoden „kreisHinzufuegen“ und „textHinzufuegen“ aufrufen können. Damit wir etwas vernünftiges sehen werden vorher noch zwei Eigenschaften vom Datenelement „m_Text“ geändert.

Statisches Element zum Zählen der Instanzen

Der zweite Teil der Aufgabe verlangt ein weiteres Datenelement in unserer Klasse „KreisText“. Damit soll gezählt werden wieviele Objekte (Instanzen) von unserer Klasse erzeugt wurden.

Ein solches Datenelement macht aber nur Sinn, wenn es *nicht* für jedes Objekt ein eigenes Datenelement gibt, viel mehr brauchen wir nur ein einzelnes Element für alle Objekte zusammen. Das wird mittels eines statischen Datenelementes erreicht. Hier der Ausschnitt aus der „KreisText“ Header-Datei, der ergänzt wurde.

```
class KreisText
{
    public:
        // Zwei Konstruktoren mit Parametern
        // Der erste nimmt zwei longs als Koordinaten
        KreisText(long x, long y, long kreisRadius,
                 ZeichenFlaeche& zeichenFlaeche);

... snip

        // statische Methode zur Abfrage der Anzahl
        // Objekte
        static unsigned long holeAnzahlObjekte()
        {
            return s_AnzahlObjekte;
        }

        // Datenelemente
    private:
        Kreis          m_Kreis;
        Text           m_Text;
        COLORREF       m_Farbe;
        Koordinaten    m_Koordinaten;

        static unsigned long s_AnzahlObjekte;
};
```

Das statische Datenelement „s_AnzahlObjekte“ dient also zum Zählen der Instanzen. Da diese Anzahl immer positiv ist, wähle ich als Datentyp *unsigned long*.

Um die Anzahl abzufragen habe ich eine statische Funktion „holeAnzahlObjekte“ definiert (direkt in der Header-Datei, also *inline*). Die Variable „s_AnzahlObjekte“ gehört nicht zu einem einzelnen Objekt, sie muss also in eine Quellcode-Datei initialisiert werden. In der „KreisText“ Quellcode-Datei (.cpp) sieht das folgendermassen aus:

```
unsigned long KreisText::s_AnzahlObjekte = 0;
```

Diesen Code setze ich meistens vor den Konstruktor der Klasse, zu der das statische Datenelement gehört. In der Konstruktoren ergänze ich nur eine Zeile, die die Anzahl um eins erhöht.

```
////////////////////////////////////
// Der Konstruktor muss eine Initialisierungsliste
// verwenden, denn die Elemente haben auch Konstruktoren
// mit Parametern
KreisText::KreisText(long x, long y, long kreisRadius,
                     ZeichenFlaeche& zeichenFlaeche)
    :m_Text(x,y), m_Kreis(x, y, kreisRadius)
{
    // Dei Konstante Weiss ist oben definiert!
    m_Text.setzeFarbe(Weiss);
    // Die Konstante InitText wurde oben definiert!
    m_Text.setzeText(InitText);
    // Mit diesem Konstruktor können wir die Elemente
    // direkt darstellen
    zeichenFlaeche.kreisHinzufuegen(m_Kreis);
    zeichenFlaeche.textHinzufuegen(m_Text);

    s_AnzahlObjekte++;
}
```

Vergiss nicht, die gleiche Zeile im zweiten Konstruktor zu ergänzen! In der main.cpp-Datei schreiben wir dann eine Funktion, die eine Zahl in einen string umwandelt. Diese können wir verwenden um die Zahl als Text auf der ZeichenFlaeche darzustellen.

```
#include <sstream>

using namespace std;

string erzeugeAnzahlText(unsigned long anzahl)
{
    stringstream stream;
    stream << "Anzahl : " << anzahl;

    string Ergebnis = stream.str();
    return Ergebnis;
}
```

Schau in meiner main.cpp-Datei nach, wie ich diese Funktion anwende.

Kopierkonstruktor für die Klasse „KreisText“ (schwer!)

Der Sinn des Kopierkonstruktors ist es ein zweites Objekt zu erzeugen, das genau wie das erste ist. Der Code um eine Kopie von einem anderen KreisText-Objekt zu erzeugen sieht so aus (zum Beispiel in der main-Funktion):

```
int main()
{
    // COM vorbereiten
    ComSystem system;

    // Die ZeichenFlaeche erzeugen mit
    // einer Breite und Hoehe
    ZeichenFlaeche dieZeichenFlaeche (400,300);
    // Hier die Farben setzen (fast weiss)
    dieZeichenFlaeche.setzeFarbe (RGB (240,240,240));

    string text ("Objekt");
    KreisText original (100,100, 50, dieZeichenFlaeche);
    original.setzeText (text);
    original.setzeKreisFarbe (100,200,101);

    // Kopie vom Objekt "original"
    // erstellen!
    KreisText kopie (original);

    getch();

    return 0;
}
```

Dieses Kopie soll nicht nur die gleichen Farben, Grössen etc. haben, sondern auch auf der gleichen „ZeichenFlaeche“ erscheinen wie das Original-Objekt. Eigenschaften vom Original-Objekt müssen von einem Objekt in das andere übertragen werden. Wir ergänzen darum alle Eigenschaften in den Datenelementen und ändern auch die meisten Funktionen in der KreisText-Klasse.

```
class KreisText
{
    public:
        // Kopierkonstruktor
        KreisText (const KreisText& c);

    ... snip

        // Datenelemente
    private:
        Kreis            m_Kreis;
        Text             m_Text;
        Koordinaten      m_Koordinaten;
        COLORREF         m_Farbe;
        string           m_Bezeichnung;
        long             m_KreisRadius;
        ZeichenFlaeche& m_ZeichenFlaeche;

        static unsigned long s_AnzahlObjekte;
};
```

Hier der Code für den Kopierkonstruktor:

```
////////////////////////////////////  
// Kopierkonstruktor  
KreisText::KreisText(const KreisText& c)  
    :m_ZeichenFlaeche(c.m_ZeichenFlaeche),  
      m_Text(c.m_Koordinaten.x, c.m_Koordinaten.y),  
      m_Kreis(c.m_Koordinaten.x, c.m_Koordinaten.y,  
c.m_KreisRadius)  
{  
    m_Text.setzeText(c.m_Bezeichnung);  
    setzeKreisFarbe(c.m_Farbe.Rot, c.m_Farbe.Gruen, c.m_Farbe.Blau);  
  
    m_ZeichenFlaeche.kreisHinzufuegen(m_Kreis);  
    m_ZeichenFlaeche.textHinzufuegen(m_Text);  
  
    s_AnzahlObjekte++;  
}
```

Beachte aber, dass alle anderen Funktionen leichte Änderungen erfahren haben.

Hole Dir die Lösung vom Netz, und versuche zu verstehen was der Code macht. Arbeite den Code Schritt für Schritt im Debugger ab!