Objektorientierte Systeme 1 - SWB2 & TIB2
Labor 6

Aufgabe 1: Grafische Objekte - Teil 6 - Exceptions

In dieser Aufgabe ergänzen Sie Ihr Programm zu den grafischen Objekten vom vorherigen Labor.

Sie haben Ihr Programm so geschrieben, dass jedes grafische Objekt eine eindeutige Identifikationsnummer besitzt, die kleiner ist als die derzeitige Maximalnummer `ObjectCounter::maxNr`. Sollte ein Objekt eine größere Nummer haben, liegt ein Fehler vor.

Sie haben Ihre Klasse `Polygonline` so geschrieben, dass immer Kopien von Punkten in eine Polygonline eingefügt werden (wenn nicht, dann haben Sie ein Problem, spätestens beim Testat). Damit gibt es keine Schleifen in einer Polygonlinie. Sollte eine Polygonlinie doch eine Schleife enthalten, endet z.B. die `print()`-Funktion nie. Probieren Sie das mit dem untenstehenden Hauptprogramm aus, nachdem Sie die notwendigen Header eingebunden haben.

```
// main.cpp zur Demonstration der vorhersehbaren Fehler

#include <iostream>
#include "Polygonline.hpp"
using namespace std;

bool debugConstructor = false;

int main() {
    // Punkt erstellen und Infos ausgeben
    const Point p;
    cout << "maxId = " << ObjectCounter::getMaxId() << endl;
    cout << "ID von p = " << p.getId() << endl;
    // ID von p unerlaubt ändern
    *(((int*)(&p))+1) = ObjectCounter::getMaxId() + 10;
    // Punkt ausgeben
    cout << "ID von p = " << p.getId() << endl;
    p.print();
    // Polygonline erstellen und ausgeben
    Polygonline pl;
    pl.addPoint(Point(1, 1));
    pl.addPoint(Point(2, 2));
    pl.addPoint(Point(3, 3));
    pl.addPoint(Point(4, 4));
    pl.print();
    // Polygonline unerlaubt ändern
    PlgElement * first = (PlgElement *)(*((int*)(&pl) + 2));
    PlgElement * last = (PlgElement *)(*((int*)(&pl) + 3));
    last->setNext(first);
```

```
    // Polygonline ausgeben
    pl.print();
    return 0;
}
```

Ziel dieser Laboraufgabe ist es, solche vorhersehbaren Fehler mit Ausnahmen abzufangen.

Hinweis: Das Hauptprogramm enthält Annahmen über das Speicherlayout der beteiligten Objekte und muss daher nicht mit jedem Compiler, Betriebssystem oder Hardware wie gewünscht ablaufen. Sollte dies bei Ihnen der Fall sein, können Sie eine andere nicht minder „schmutzige“ Lösung probieren: setzen Sie die Attribute `id`, `first` und `last` in den jeweiligen Klassen auf **`public`**, dann ändern Sie das Hauptprogramm so ab, dass diese Attribute direkt (ohne die zweifelhaften Casts) gelesen bzw. geändert werden. Eine weitere Lösung, die etwas mehr Änderungsaufwand erfordert, besteht darin, dass Sie für den Zugriff auf die genannten Attribute set- bzw. get-Methoden einführen. Wichtig ist nur, dass die gewünschten Effekte bzw. Exceptions auftreten. Das ist ja das Ziel der Aufgabe. Wie man dazu kommt ist zweitrangig.

Schreiben Sie eine Klasse `GraphException` in der Klasse `DrawingObject`. Die Klasse hat eine konstante Instanzvariable `id`, die die Nummer des grafischen Objektes angibt, bei dem ein Fehler festgestellt wurde. Schreiben Sie einen geeigneten Konstruktor. Definieren Sie dann in der Klasse `DrawingObject` die Klasse `IdTooHigh` als abgeleitete Klasse der Klasse `GraphException`. Schreiben Sie einen geeigneten Konstruktor sowie eine Methode `getId()`.

Schreiben Sie in der Klasse `DrawingObject` eine Methode **`void`** `check()` **`const`**, die eine Ausnahme der Klasse `IdTooHigh` wirft, wenn die Identifikationsnummer des Objektes größer als `ObjectCounter::maxId` ist. Die Methode wird passenderweise innerhalb der Methode `getId()` aufgerufen.

Schreiben Sie in der Klasse `Polygonline` die Klasse `LoopInLine` mit einem geeigneten Konstruktor. Diese Klasse ist von `GraphException` abgeleitet. Ändern Sie die `print()`-Methode von der Klasse `Polygonline` so ab, dass eine Ausnahme `LoopInLine` geworfen wird, wenn die Methode versucht, denselben Punkt zweimal auszugeben.

Ändern Sie die oben gegebene Hauptfunktion ab, so dass die beiden speziellen Ausnahmen als auch alle Ausnahmen vom Typ `GraphException` aufgefangen werden und testen Sie es.

Aufgabe 2: Grafische Objekte - Teil 7 - Ausgabe in Datei

Nutzen Sie die Klassen der grafischen Objekte aus Labor 5.

Schreiben Sie ein Hauptprogramm, mit dem Sie Strings einlesen und damit Circle-Objekte erstellen. Die Anzahl der einzulesenden Strings soll benutzergesteuert sein. Definieren Sie daher ein sinnvolles Abbruchkriterium für die Einleseschleife.

Öffnen Sie eine Binärdatei und schreiben Sie jedes Circle-Objekt binär sofort nach dem Einlesen hinein. Jedes Circle-Objekt belegt einen Datensatz. Zählen Sie mit, wie viele Objekte Sie abspeichern und schreiben Sie zum Schluss diese Anzahl ebenfalls binär vorne in die Datei hinein. D.h. am Beginn der Datei müssen Sie Platz für die Anzahl lassen, die Sie erst am Ende kennen.

Aufgabe 3: Grafische Objekte - Teil 8 - Einlesen von Datei

Schreiben Sie ein weiteres Hauptprogramm (in einem getrennten Projekt), mit dem Sie die in der vorherigen Aufgabe geschriebene Datei einlesen. Lesen Sie zuerst die Anzahl aus dem Anfang der Datei ein. Deklarieren Sie dann ein dynamisches Array der notwendigen Größe für die einzulesenden Circle-Objekte. Lesen Sie alle Circle-Objekte aus der Datei ein und speichern Sie die Objekte in dem Array.

Geben Sie dann die Objekte im Array über die virtuelle Methode `print()` aus, um die Korrektheit und Vollständigkeit des Schreib- und Lesevorgangs zu prüfen.