UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

6. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

Klassen und Vererbung / Exceptions

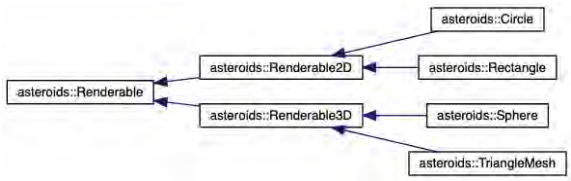
In dieser Übung setzen wir und mit Vererbung in C++ auseinander. Dazu erweitern wir die Musterlösung des letzten Aufgabenblattes um Funktionalitäten zur Fehlerbehandlung und Klassen zur Darstellung von 2D-Objekten.

Aufgabe 1: Exception-Handling für Mathe-Klassen (30 Punkte)

Erweitern Sie die Klassen `Vector-`, `Matrix-` und `Quaternion`-Klassen um Exceptions. Implementieren Sie dazu eine Klasse `BaseException` und leiten Sie davon die Klassen `OutOfBoundsException` und `DivisionByZeroException` ab. In der letzten Übung wurde vom Indexoperator der Matrix-Klasse ein Zeiger auf die entsprechende Speicherstelle zurückgegeben. Um eine ordentliche Fehlerbehandlung zu realisieren, soll der Indexoperator nun ein Proxyobjekt zurückgeben, das wiederum einen eigenen Indexoperator implementiert. Fangen Sie nun Fehler beim Zugriff auf Matrixelemente ab, indem Sie eine `OutOfBoundsException` werfen, sobald auf einen ungültigen Index der Matrix zugegriffen wird. Werfen Sie eine `DivisionByZeroException`, sobald irgendwo eine Division durch 0 stattfindet. Ergänzen Sie den Code der Musterlösung entsprechend und integrieren Sie Code zur Fehlerbehandlung, falls an einer Stelle eine Exception auftreten kann.

Aufgabe 2: Klassenhierarchie für grafische Objekte (70 Punkte)

Um das Zeichnen von Objekten in unserem Spiel zu strukturieren soll folgende Klassenhierarchie implementiert werden:



Die abstrakte Klasse `Renderable` stellt dabei das Interface für alle darstellbaren Objekte dar. Sie stellt die Methode `render()` zur Verfügung, die von allen Unterklassen passend Implementiert werden muss. Die Klasse `Renderable3D` stellt die Basisklasse für alle Objekte dar, die im 3D-Raum gerendert werden. Die Klassen `TriangleMesh` ist die alte `Model`-Klasse, die das Raumschiff rendert. Die Klasse `Sphere` stellt Funktionalität zum Rendern einer Kugel im 3D-Raum zur Verfügung. Sie hat folgende (minmal) Signatur:

```
class Sphere: public Renderable3D
{
public:
    /**
     * @brief Construct a new Sphere object
     *
     * @param position      Initial position of the sphere
     * @param radius        Radius
     * @param numSides      Number of horizontal intersections
     * @param numStack      Number of vertical intersections
     */
    Sphere( const Vector& position, float radius, int numSides = 10, int numStack = 10);

    /// Renders the sphere at the given position
    virtual void render();
  ...

};
```

Der Konstruktor initialisiert eine Kugel an der gegebenen Position. Zum Rendern wird die Kugel entlang des Äquators in `numSides` Teile unterteilt, vertikal in `numStack` Stücke. Die Parameter bestimmen also, wie genau eine Kugel beim Rendern approximiert wird. Der OpenGL-Code zum Rendern der Kugel wird Ihnen in StudIP zur Verfügung gestellt. Die Klasse Renderable2D ist die Basisklasse für Objekte, die in 2D-Bildschirmkoordinaten

gezeichnet werden. Dazu muss die aktuelle Dimensionierung des Fensters bekannt sein. Entsprechend wird intern ein Zeiger auf eine `MainWindow`-Instanz gespeichert. Sie hat folgende Minimalsignatur:

```
class Renderable2D : public Renderable
{
public:

    /// Constructor
    Renderable2D(MainWindow* mainWindow);

    /// Renders the object
    virtual void render() = 0;

    /// Sets the current rendering color
    void setColor(float r, float g, float b);

          …

};
```

Um Objekte in 2D-Koordinaten rendern zu können, ist es notwendig, in OpenGL mehrere Operationen durchzuführen:

1.) Sichern der aktuellen Modelview-Matrix und Reset für die aktuellen Operationen:

```
// Enter modelview mode and save current view
// matrix. Set transformation to indentity to
// 'undo' current look at transformation
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
```

2.) Erstellung einer Orthogonalprojektionsmatrix, die auf die Größe des aktuellen Fensters angepasst ist:

```
// Enter projection mode and set ortho projection
// according to current window size
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
glOrtho(0.0f, m_mainWindow->width(), m_mainWindow->height(), 0.0f, -10.0f, 10.0f);
```

3.) Wiederherstellung der 3D-Darstellung

```
// Delete current ortho projection, enter model
// view mode and restore previous look at matrix
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
glPopMatrix();
```

Nach den Schritten 1.) und 2.) können 2D-Vertices mittels `glVertex2f` definiert werden. Leiten Sie von der Klasse `Renderable2D` die Klassen `Rectangle` und `Circle` ab und implementieren sie die passenden Render-Methoden. Erläutern Sie, warum es Sinn macht, dass die Klasse `Renderable2D`, obwohl Sie eine abstrakte Klasse ist, einen Konstruktor zur Verfügung stellt.

Der entsprechende OpenGL-Code zum Rendern von Rechtecken und Kreisen wird ihnen ebenfalls als Snippet in StudIP zur Verfügung gestellt. Implementieren Sie die beschriebene Klassenstruktur. Sie können Ihren Code testen, indem Sie die markierten Stellen in der Methode `mainLoop()` der Klasse `MainWindow` wieder einkommentieren. Lagern Sie bei der Implementierung gemeinsame Funktionalität so weit wie möglich in die entsprechenden Oberklassen aus. Vermeiden Sie jegliche Art von Code-Verdoppelung!

Hinweise zur Bearbeitung der Aufgabe:

Die hier vorgestellten Signaturen stellen nur Ausschnitte mit den minimal benötigen Elementen der zu implementierenden Klassen dar! Um die Anforderungen zu erfüllen, müssen Sie die Klassen gegebenenfalls um weitere Hilfsfunktionen und interne Felder erweitern. Schränken Sie dabei deren Sichtbarkeit so weit wie möglich ein. Überlegen Sie sich, welche Klassenelemente sie als `virtual` deklarieren sollten und welche nicht.

Abgabe:

Checken Sie Ihre Lösung des Aufgabenblattes bis Montag den 03.12.2018, 8:00 Uhr in den Master-Branch des git-Repositorys Ihrer Gruppe ein. Bringen Sie pro Gruppe einen Ausdruck des von Ihnen erstellten Codes mit.