Einführung in C++ – Übung 8
Testatgruppe A (Isaak)

Rasmus Diederichsen

7. Dezember 2014

Aufgabe 8.1 Auslagern von Funktionalität in Oberklassen

Die `Renderable`-Klasse kann abstrakt sein, da sie nur eine Methode enthält, die Subklassen überschreiben müssen (sollten), damit etwas passiert. Es ist nicht sinnvoll, die Funktionalität in der Oberklasse zu implementieren, da sich die `render()`-Methoden zwischen `TriangleMesh` und `TexturedMesh` unterscheiden.

Listing 1: Renderable.hpp

```
/**
 * @file Renderable.hpp
 * @author Rasmus Diederichsen (rdiederichse@uos.de)
 * @version 04.12.2014
 */

#ifndef RENDERABLE_KG6LA0OJ

#define RENDERABLE_KG6LA0OJ

namespace asteroids {

   /**
    * @class Renderable
    * @brief Abstract class to represent something which can be
        rendered on screen.
    */
   class Renderable {

      /**
       * @brief Render this object (pure virtual)
       */
      virtual void render () = 0;
   };
}

#endif /* end of include guard: RENDERABLE_KG6LA0OJ */
```

Hinweis: Die Methoden dieser Klasse sind in `TriangleMesh.cpp` implementiert, diese fehlten initial. Auch wurde in allen Klassen die Acquise des Singletons auf Pointer umgestellt.

Listing 2: Transformable.hpp

```
#ifndef TRANSFORMABLE_DILCYMZ4

#define TRANSFORMABLE_DILCYMZ4

#include "Renderable.hpp"
#ifdef __APPLE__
#include <OpenGL/gl.h>
#else
#include <GL/gl.h>
#endif

namespace asteroids

{
   /**
    * @class Transformable
    * @brief Abstract base class for <tt>Mesh</tt>es which can be
       moved &
    * rotated.
    */
   class Transformable : public Renderable
   {
      public:
         /**
          * @brief   Rotate the Transformable
          * @param axis axis of rotation
          * @param speed speed of rotation
          */
         virtual void rotate(int axis, float speed) = 0;

         /**
          * @brief   Move the Transformable along acis
          * @param axis axis to move along
          * @param speed speed of movement
          */
         virtual void move(int axis, float speed) = 0;

      protected:
         /**
          * @brief Array containing the transformation matrix.
          */
         float m_transformation[16];

         /**
          * @brief Method to compute the transformation matrix.
          */
         virtual void computeMatrix() = 0;
   };
}

#endif /* end of include guard: TRANSFORMABLE_DILCYMZ4 */
```

Das `FixedObject` ist leer, also nur ein Markerinterface.

Listing 3: FixedObject.hpp

```
#ifndef FIXEDOBJECT_4ST6ZIUC

#define FIXEDOBJECT_4ST6ZIUC

#include "Renderable.hpp"

namespace asteroids {

   /**
    * @class FixedObject
    * @brief Represents something which cannot be moved or rotated.
    */
   class FixedObject : public Renderable { };
}

#endif /* end of include guard: FIXEDOBJECT_4ST6ZIUC */
```

Aufgabe 8.2 Erweiterungen für neue Dateiformate

Listing 4: TriangleMeshFactory.hpp

```
/**
 * @brief Contains Factory for mesh generation.
 *
 * @file TriangleMeshFactory.hpp
 * @author rdiederichse@uos.de
 */
#ifndef TRIANGLEMESHFACTORY_H

#define TRIANGLEMESHFACTORY_H

#include "rendering/TriangleMesh.hpp"

namespace asteroids
{
   /**
    * @class TriangleMeshFactory
    * @brief Singleton Factory class to encapsulate parsing meshes
        from different file
    * types.
    */
   class TriangleMeshFactory
   {
      public:
         /**
          * @brief Returns a pointer to a mesh parsed from a given
              file. Format
          * is recognized by extension.
          *
          * @param filename The file containing the mesh definition
              .
```

```
28              * @return A pointer to the parsed mesh
29              */
30             TriangleMesh* getMesh(const std::string &filename) const;
31
32             /**
33              * @brief Method to acquire the singleton instance
34              * @return The singleton.
35              */
36             static TriangleMeshFactory* instance();
37
38         private:
39             /**
40              * Empty default constructor. Does nothing.
41              */
42             TriangleMeshFactory();
43
44             /**
45              * The singleton instance.
46              */
47             static TriangleMeshFactory* instance_ptr;
48
49             /**
50              * Copy constructor. Does nothing.
51              */
52             TriangleMeshFactory(const TriangleMeshFactory& f) {};
53
54             /**
55              * Assignment operator. Does nothing.
56              */
57             TriangleMeshFactory& operator=(const TriangleMesh& f) {};
58     };
59 } /* namespace asteroids */
60 #endif /* end of include guard: TRIANGLEMESHFACTORY_H */
```

Listing 5: TriangleMeshFactory.cpp

```
1  /**
2   * @brief TriangleMeshFactory implementation.
3   * @file TriangleMeshFactory.cpp
4   * @author rdiederichse@uos.de
5   */
6  #include "io/TriangleMeshFactory.hpp"
7  #include "ReadPLY.hpp"
8  #include "Read3DS.hpp"
9  #include "MeshReader.hpp"
10
11 using std::string;
12
13 namespace asteroids
14 {
15
16     TriangleMeshFactory::TriangleMeshFactory() {}
17     TriangleMeshFactory* TriangleMeshFactory::instance_ptr = NULL;
18
19     TriangleMeshFactory* TriangleMeshFactory::instance()
20     {
21         if (instance_ptr != NULL) return instance_ptr;
22         else return (instance_ptr = new TriangleMeshFactory);
```

```
23      }
24
25      TriangleMesh* TriangleMeshFactory::getMesh(const string &
            filename) const
26      {
27          unsigned found = filename.find_last_of("/\\");
28          string basePath = filename.substr(0, found+1);
29          TextureFactory::setBasePath(basePath);
30
31          int pos;
32          MeshReader *reader;
33          if ((pos = filename.find(".ply")) == (filename.length() - 4))
                // .ply extension
34          {
35              reader = new ReadPLY(filename);
36              return reader->getMesh();
37          } else if ((pos = filename.find(".3ds")) == (filename.length
                () - 4)) // .3ds extension
38          {
39              reader = new Read3DS(filename);
40              return reader->getMesh();
41          } else  // error
42          {
43              int i = filename.find_last_of(".");
44              std::cerr << "TriangleMeshFactory␣Error:␣File␣" <<
                    filename << (i == string::npos ? "␣without␣extension"
                    : "␣of␣type␣" + filename.substr(i))
45                  << "␣not␣readable.␣" << std::endl;
46              return NULL;
47          }
48      }
49
50  } /* namespace asteroids */
```

Listing 6: TextureFactory.hpp

```
1   #ifndef TEXTUREFACTORY_H
2
3   #define TEXTUREFACTORY_H
4
5   #include "rendering/Texture.hpp"
6
7   namespace asteroids
8   {
9       class TextureFactory
10      {
11          public:
12              /**
13               * @brief Returns a pointer to a textured mesh parsed from
                     a given file. Format
14               * is recognized by extension.
15               *
16               * @param filename The file containing the mesh definition
                     .
17               * @return A pointer to the parsed mesh
18               */
19              Texture* getTexture(const std::string &filename) const;
20
```

```
21          /**
22           * @brief Method to acquire the singleton instance
23           * @return The singleton.
24           */
25          static TextureFactory* instance();
26
27          /**
28           * Set the base path relative to which textures will be
                 loaded.
29           * @param basepath The path relative to which textures are
                 loaded.
30           */
31          static void setBasePath(std::string basepath);
32
33      private:
34          /**
35           * Empty default constructor. Does nothing.
36           */
37          TextureFactory();
38
39          /**
40           * The singleton instance.
41           */
42          static TextureFactory* instance_ptr;
43
44          /**
45           * Copy constructor. Does nothing.
46           */
47          TextureFactory(const TextureFactory& f) {};
48
49          /**
50           * Assignment operator. Does nothing.
51           */
52          TextureFactory& operator=(const TextureFactory& f) {};
53
54          /**
55           * @brief The base path.
56           */
57          static std::string basepath;
58  };
59  } /* namespace asteroids */
60
61  #endif /* end of include guard: TEXTUREFACTORY_H */
```

Listing 7: TextureFactory.cpp

```
1   /**
2    * @file TextureFactory.cpp
3    * @author Rasmus Diederichsen (rdiederichse@uos.de)
4    * @version 03.12.2014
5    */
6
7   #include "TextureFactory.hpp"
8   #include "BitmapReader.hpp"
9   #include "ReadPPM.hpp"
10  #include "ReadJPG.hpp"
11  #include "ReadTGA.hpp"
12  #include <iostream>
```

```
13 #include <cstddef>
14
15 namespace asteroids
16 {
17
18    TextureFactory::TextureFactory() {}
19    TextureFactory* TextureFactory::instance_ptr = NULL;
20    std::string TextureFactory::basepath = "";
21
22    TextureFactory* TextureFactory::instance()
23    {
24       if (instance_ptr != NULL) return instance_ptr;
25       else return (instance_ptr = new TextureFactory);
26    }
27
28    Texture* TextureFactory::getTexture(const string &filename)
          const
29    {
30       int pos;
31       BitmapReader *reader;
32       if ((pos = filename.find(".ppm")) == (filename.length() - 4))
             // .ppm extension
33       {
34          reader = new ReadPPM(basepath + filename);
35          return new Texture(reader->getPixels(), reader->getWidth()
                , reader->getHeight());
36       } else if ((pos = filename.find(".jpg")) == (filename.length
             () - 4)) // .jpg extension
37       {
38          reader = new ReadJPG(basepath + filename);
39          return new Texture(reader->getPixels(), reader->getWidth()
                , reader->getHeight());
40       } else if ((pos = filename.find(".tga")) == (filename.length
             () - 4))
41       {
42          reader = new ReadTGA(basepath + filename);
43          return new Texture(reader->getPixels(), reader->getWidth()
                , reader->getHeight());
44       } else  // error
45       {
46          int i = filename.find_last_of(".");
47          std::cerr << "TextureFactory␣Error:␣File␣" << filename
48             << (i == string::npos ? "␣without␣extension" : "␣of␣
                   type␣" + filename.substr(i))
49             << "␣not␣readable.␣" << std::endl;
50          return NULL;
51       }
52    }
53
54    void TextureFactory::setBasePath(string basepath)
55    {
56       TextureFactory::basepath = basepath;
57    }
58
59
60 } /* namespace asteroids */
```

Listing 8: ReadPPM.hpp

```
/**
 * @file ReadPPM.hpp
 * @author Rasmus Diederichsen (rdiederichse@uos.de)
 * @version 02.12.2014
 */

#ifndef READPPM_H

#define READPPM_H

#include "BitmapReader.hpp"
#include <fstream>

using std::ifstream;

namespace asteroids
{
   /**
    * @enum PPMTYPE
    * @brief Constants for PPM types.
    */
   enum PPMTYPE {
      P3, ///< Ascii file
      P6,  ///< Binary file
      UNDEF ///< Unknown type (should lead to error)
   };

   /**
    * @class ReadPPM
    * @brief A reader for ppm images.
    */
   class ReadPPM : public BitmapReader
   {
      private:
         /**
          * @brief Reads linewise from a ppm file as long as the
              current line starts
          * with a '#'.
          *
          * After the method has run, the stream's current line
              will
          * be the first non-comment line after the position for
              which the
          * method was called.
          * @param stream The input file stream to read from.
          * @param buffer Char buffer to place the bytes or chars
              in.
          * @param bufsize The length of the buffer.
          */
         void readFirstNonCommentLine(ifstream& stream, char*
             buffer, const int bufsize);

         /**
          * @brief Parses a ppm header. Sets the members <tt>
              m_width</tt>,
          * <tt>m_height</tt> and reads the magic number.
```

```
51            * @param filename The name of the file.
52            * @return The type of the file (binary or ascii). One of
53            * <tt>PPMTYPE</tt>
54            * @see ReadPPM::PPMTYPE
55            */
56           PPMTYPE readHeader(const std::string& filename);
57
58           /**
59            * The position right after the header. Seeking to this
                 position and
60            * then reading will read everything except the header.
61            */
62           ifstream::pos_type end_header;
63
64           /**
65            * @brief Read an ascii ppm file.
66            * @param filename The name of the file.
67            */
68           void readP3(const std::string& filename);
69
70           /**
71            * @brief Read a binary ppm file.
72            * @param filename The name of the file.
73            */
74           void readP6(const std::string& filename);
75       public:
76
77           /**
78            * @brief Construct a PPM reader to read from a given file
                 .
79            * @param filename The name of the file.
80            */
81           ReadPPM(const std::string& filename);
82
83           /**
84            * @brief Empty destructor. Deallocation of the image
                 buffer must be
85            * taken are of by the client.
86            */
87           ~ReadPPM();
88   };
89 } /* namespace asteroids */
90
91 #endif /* end of include guard: READPPM_H */
```

Listing 9: ReadPPM.cpp

```
1  /**
2   * @file ReadPPM.cpp
3   * @author Rasmus Diederichsen (rdiederichse@uos.de)
4   * @version 02.12.2014
5   */
6
7  #include "ReadPPM.hpp"
8  #include <sstream>
9  #include <iostream>
10
11 using std::cout;
```

```
12 using std::cerr;
13 using std::endl;
14 using std::string;
15 using std::stringstream;
16 using std::ifstream;
17 using std::ios;
18
19 namespace asteroids
20 {
21    ReadPPM::ReadPPM(const std::string& filename)
22    {
23       cout << "ReadPPM: reading ppm file " << filename << endl;
24       m_pixels = NULL; // in case shit hits the fan.
25       m_height = m_width = 0;
26       PPMTYPE type = readHeader(filename); // parse header and
             recognise file type
27       // allocate only if needed.
28       if (type != UNDEF) m_pixels = new unsigned char[m_width *
             m_height * 3];
29
30       switch (type)
31       {
32          case P3:
33             readP3(filename);
34             break;
35          case P6:
36             readP6(filename);
37             break;
38          default:
39             cerr << "ReadPPM: Error. Unknown PPM format." << endl;
40             break;
41       }
42    }
43
44    void ReadPPM::readFirstNonCommentLine(ifstream& stream, char*
          buffer, const int bufsize)
45    {
46       do { // read at least one line
47          stream.getline(buffer, bufsize);
48       } while(stream.good() && (buffer[0] == '#')); // continue
             while comment
49       // now we have read the first non-comment line into the
             buffer.
50    }
51
52    PPMTYPE ReadPPM::readHeader(const string& filename)
53    {
54       ifstream file(filename.c_str());
55       if (file.good())
56       {
57          /* ATTENTION. THIS ERRONEOUSLY ASSUMES THAT EVERYTHING IS
                SEPARATED BY
58             NEWLINES, EXCEPT THE DIMENSIONS. NEED TO FIX THIS. */
59          const int bufsize = 70; // Acc. to spec, lines should not
                be longer
60          char buffer[bufsize]; // buffer to hold the line
61          readFirstNonCommentLine(file, buffer, bufsize); // fill it
```

```
62          string magic_number(buffer); // first line must contain
                magic number
63          readFirstNonCommentLine(file, buffer, bufsize);
64          string line(buffer); // next one must contain dimensions
65          stringstream ss(line); ss >> m_width >> m_height; // get
                them out
66          readFirstNonCommentLine(file, buffer, bufsize);
67          file.getline(buffer, bufsize); // skip the max color value
                . Assume it's 255..
68          end_header = file.tellg();
69          file.close();
70
71          cout << "ReadPPM: Parsed header of " << filename << ". 
                width=" << m_width
72             <<" height=" << m_height << " magic number is "
73             << magic_number  << endl;
74
75          if (magic_number == "P3") return P3;
76          else if (magic_number == "P6") return P6;
77       }  return UNDEF;
78    }
79
80    void ReadPPM::readP3(const string& filename)
81    {
82       cout << "ReadPPM: Reading ascii file " << filename << endl;
83       ifstream file(filename.c_str());
84       if (file.good())
85       {
86          file.seekg(end_header); // go back to where we left off
87          // the stream splits on whitespace so the chars can be
                read one after
88          // the other without hassle.
89          for(int i = 0; i < m_width * m_height * 3; i++)
90          {
91             int c;
92             file >> c; // we need to map an ascii number symbol (or
                     more) to the
93                        // numerical value, not just use the symbols
                             itself
94             m_pixels[i] = (char)c;
95          }
96          file.close();
97       } else { cerr << "ReadPPM: Error, P3 file not good." << endl;
            } // TODO: Detailed error
98    }
99
100   void ReadPPM::readP6(const string& filename)
101   {
102      cout << "ReadPPM: Reading binary file " << filename << endl;
103      ifstream file(filename.c_str(), ios::binary); // open in
             binary mode
104      if (file.good())
105      {
106         file.seekg(end_header); // go back to where we left off
107         file.read((char *)m_pixels,m_width * m_height * 3); // not
                sure if good. dump whole binary blob into array.
108         file.close();
```

```
109         } else { cerr << "ReadPPM: Error, P6 file not good." << endl;
                } // TODO: Detailed error
110     }
111
112     ReadPPM::~ReadPPM() { }
113
114 } /* namespace asteroids */
```