SCHETT MATTHIAS

SEN-ÜBUNG 06

Inhaltsverzeichnis

Aufgabe 1

Lösungsidee

Die Realisierung erfolgt mittels der Klassen Stock[1] und StockMarket[2]. Die Stock Klasse besitzt für jeden Member[3] eine Getter und eine Setter Methode. Zusätzlich exisitiert noch die calcNewValuesOnChangeRate(double changeRate); Methode. Mit deren Hilfe werden alle Memberwerte neu berrechnet. Die StockMarket Klasse speichert ihre Aktien in einer StockCollection[4], darin befinden sich StockEntries[5]. Um neue Aktien hinzuzufügen, steht die readStocks(std::istream &istream) Methode bereit, mit deren Hilfe die Scanner Klasse Aktien einliest. Bei einem Fehler im Input Stream wird eine exception mit <Unknown format> geworfen. Die Simulation erfolgt über die Nicht Klassenfunktion simulateStockMarket(StockMarket &market, size_t numOfDaysToSimulate);[6] Die Ausgabe erfolgt schließlich der vorgegebenen Tabellenform.

Der Quellcode findet sich im Anhang ab Aufgabe 1

[1] Zur Verwaltung einer einzelnen Aktie mit Namen und den benötigten Werten

[2] Zur Verwaltung einer Sammlung von Aktien

[3] std::string mStockName
double mAcutalSharePrice;
double mDayBeforeSharePrice;
double mHighestSharePrice;
double mLowestSharePrice;
double mStockChangeRate;

[4] Aktuell ein typedef von std::vector

[5] typedef der Stock Klasse

[6] Ruft in einer Iteration für jede enthaltene Stock in einem StockMarket die calcNewValuesOnChangeRate, numOfDaysToSimulate mal auf

Testfälle

Listing 1: Input für Testfall 1

```
1 "Andritz AG": 68.00
2 "Zumtobel": 24.61
3 "VOEST Alpine": 33.88
4 "Telekom Austria": 10.34
5 "OMV": 31.90
```

Listing 2: Test Ausgabe

```
Table name sorted
-------------------------------------------------------------------------------
Aktien              (+/-) Proz    Aktuell     Vortag       Hoch        Tief
-------------------------------------------------------------------------------
Andritz AG              0.98     172.58     168.83     172.58      60.05
OMV                     0.99      11.87      11.70      38.93      11.10
Telekom Austria         0.97       9.12       8.87      15.96       7.02
VOEST Alpine            1.02      47.03      47.75      50.63      28.92
Zumtobel                0.97      15.51      14.99      26.51      10.13



Table highest value sorted
-------------------------------------------------------------------------------
Aktien              (+/-) Proz    Aktuell     Vortag       Hoch        Tief
-------------------------------------------------------------------------------
Andritz AG              0.98     172.58     168.83     172.58      60.05
VOEST Alpine            1.02      47.03      47.75      50.63      28.92
Zumtobel                0.97      15.51      14.99      26.51      10.13
OMV                     0.99      11.87      11.70      38.93      11.10
Telekom Austria         0.97       9.12       8.87      15.96       7.02



Table lowest value sorted
-------------------------------------------------------------------------------
Aktien              (+/-) Proz    Aktuell     Vortag       Hoch        Tief
-------------------------------------------------------------------------------
Andritz AG              0.98     172.58     168.83     172.58      60.05
VOEST Alpine            1.02      47.03      47.75      50.63      28.92
OMV                     0.99      11.87      11.70      38.93      11.10
Zumtobel                0.97      15.51      14.99      26.51      10.13
Telekom Austria         0.97       9.12       8.87      15.96       7.02



Table highest diff value sorted
-------------------------------------------------------------------------------
Aktien              (+/-) Proz    Aktuell     Vortag       Hoch        Tief
-------------------------------------------------------------------------------
Andritz AG              0.98     172.58     168.83     172.58      60.05
OMV                     0.99      11.87      11.70      38.93      11.10
VOEST Alpine            1.02      47.03      47.75      50.63      28.92
Zumtobel                0.97      15.51      14.99      26.51      10.13
Telekom Austria         0.97       9.12       8.87      15.96       7.02
```

Aufgabe 2

Lösungsidee

Die Werte werden mittels der Extract Funktion in zwei Iterationen extrahiert, die erste geht über alle Aktien und die zweite über alle gepseicherten Werte in der Collection in der Stock Klasse. Die Ausgabe erfolgt anschließend in einer Iteration über die neue Collection und durch Ausgae des Aktiennamens und der Anzahl an Elementen innerhalb der gefilterten Werte Collection. Der Quellcode wurde aus Aufgabe 1 übernommen und mit den Methoden

- StockCollection Extract(StockCollection::iterator begin, StockCollection::iterator end);
- void printExtracted(StockCollection extractedValues, std::ostream &os);
- void extractAndPrint(std::ostream &os);

erweitert.

Daher befindet sich der Quellcode auch hier im Anhang ab Aufgabe 1

Testfälle

Der Input ist der selbe wie in Abschnitt Testfälle von Aufgabe 1.

Listing 3: Test Ausgabe

```
1 Andritz AG: 333
2 VOEST Alpine: 316
3 Telekom Austria: 285
4 OMV: 54
5 Zumtobel: 19
```

Anhang A
Aufgabe 1

Listing A.1: Header der Stock Klasse

```
1  ////////////////////////////////////
2  // Workfile     : Stock.h
3  // Author       : Matthias Schett
4  // Date         : 27-04-2013
5  // Description  : Stock management
6  // Remarks      : -
7  // Revision     : 0
8  ////////////////////////////////////
9
10 #ifndef STOCK_H
11 #define STOCK_H
12
13
14 #include <string>
15 #include <vector>
16 class Stock {
17
18 private:
19     std::string mStockName;
20     double mAcutalSharePrice;
21     double mDayBeforeSharePrice;
22     double mHighestSharePrice;
23     double mLowestSharePrice;
24     double mStockChangeRate;
25     std::vector<double> mSharePriceCollection;
26
27 public:
28     // Ctr
29     Stock(std::string const &stockName, double actualPrice);
30     // Ctr for Extract Method
31     Stock(std::string const &stockName, std::vector<double>
           sharePriceCollection);
32     // Dtr
33     ~Stock();
34
```

```
35      // Getters
36      std::string const &getStockName() const;
37      double getActualSharePrice() const;
38      double getDayBeforeSharePrice() const;
39      double getHighestSharePrice() const;
40      double getLowestSharePrice() const;
41      double getStockChangeRate() const;
42      std::vector<double> getSharePriceCollection() const;
43
44      // Setters
45      void setStockName(std::string const &stockName);
46      void setActualSharePrice(double actualSharePrice);
47      void setDayBeforeSharePrice(double dayBeforeSharePrice);
48      void setHighestSharePrice(double highestSharePrice);
49      void setLowestSharePrice(double lowestSharePrice);
50      void setStockChangeRate(double stockChangeRate);
51
52      //**********************************
53      // Method:     calcNewValuesOnChangeRate
54      // FullName:   Stock::calcNewValuesOnChangeRate
55      // Access:     public
56      // Returns:    void
57      // Qualifier:
58      // Parameter: double changeRate
59      // Calculates the new stock values based on the given change rate
60      //**********************************
61      void calcNewValuesOnChangeRate(double changeRate);
62
63  };
64
65  #endif
```

Listing A.2: Implementierung der Stock Klasse

```
1   ////////////////////////////////////////
2   // Workfile     : Stock.cpp
3   // Author       : Matthias Schett
4   // Date         : 27-04-2013
5   // Description  : Stock management
6   // Remarks      : -
7   // Revision     : 0
8   ////////////////////////////////////////
9
10  #include "Stock.h"
11
12
13  Stock::Stock(std::string const &stockName, double actualPrice) : mStockName(
        stockName), mAcutalSharePrice(actualPrice), mDayBeforeSharePrice(0),
14      mHighestSharePrice(0), mLowestSharePrice(0), mStockChangeRate(0),
            mSharePriceCollection(1, mAcutalSharePrice) {
```

```
15 }
16
17
18 Stock::Stock(std::string const &stockName, std::vector<double>
       sharePriceCollection) : mStockName(stockName), mAcutalSharePrice(
       sharePriceCollection.front()), mDayBeforeSharePrice(0),
19     mHighestSharePrice(0), mLowestSharePrice(0), mStockChangeRate(0),
           mSharePriceCollection(sharePriceCollection){
20
21 }
22
23 Stock::~Stock(void)
24 {
25 }
26
27 // Getters
28 std::string const &Stock::getStockName() const{
29     return mStockName;
30 }
31
32 double Stock::getActualSharePrice() const{
33     return mAcutalSharePrice;
34 }
35
36 double Stock::getDayBeforeSharePrice() const{
37     return mDayBeforeSharePrice;
38 }
39
40 double Stock::getHighestSharePrice() const{
41     return mHighestSharePrice;
42 }
43
44 double Stock::getLowestSharePrice() const{
45     return mLowestSharePrice;
46 }
47
48 double Stock::getStockChangeRate() const{
49     return mStockChangeRate;
50 }
51
52 // Setters
53
54 void Stock::setStockName(std::string const &stockName){
55     mStockName = stockName;
56 }
57
58 void Stock::setActualSharePrice(double actualSharePrice){
59     mAcutalSharePrice = actualSharePrice;
60 }
```

```
61
62 void Stock::setDayBeforeSharePrice(double dayBeforeSharePrice){
63     mDayBeforeSharePrice = dayBeforeSharePrice;
64 }
65
66 void Stock::setHighestSharePrice(double highestSharePrice){
67     mHighestSharePrice = highestSharePrice;
68 }
69
70 void Stock::setLowestSharePrice(double lowestSharePrice){
71     mLowestSharePrice = lowestSharePrice;
72 }
73
74 void Stock::setStockChangeRate(double stockChangeRate){
75     mStockChangeRate = stockChangeRate;
76 }
77
78 void Stock::calcNewValuesOnChangeRate( double changeRate ) {
79
80     setDayBeforeSharePrice(getActualSharePrice());
81     setActualSharePrice(getActualSharePrice() + (getActualSharePrice() *
          changeRate));
82
83     mSharePriceCollection.push_back(getActualSharePrice());
84
85     if(getHighestSharePrice() < getActualSharePrice()){
86         setHighestSharePrice(getActualSharePrice());
87     }
88
89     if(getLowestSharePrice() > getActualSharePrice() || getLowestSharePrice()
          == 0){
90         setLowestSharePrice(getActualSharePrice());
91     }
92
93     setStockChangeRate(getDayBeforeSharePrice() / getActualSharePrice());
94
95 }
96
97 std::vector<double> Stock::getSharePriceCollection() const {
98     return mSharePriceCollection;
99 }
```

Listing A.3: Header der StockMarket Klasse

```
1 ////////////////////////////////////
2 // Workfile    : StockMarket.h
3 // Author      : Matthias Schett
4 // Date        : 27-04-2013
5 // Description : Stock management
6 // Remarks     : -
```

```
7  // Revision     : 0
8  ////////////////////////////////////
9
10 #ifndef STOCKMARKET_H
11 #define STOCKMARKET_H
12
13
14 #include "Stock.h"
15 #include <vector>
16 #include <string>
17 #include <istream>
18 #include <ostream>
19
20
21 size_t const colWidthNum = 10;
22 size_t const precision = 2;
23 size_t const colSpacing = 4;
24
25 // Defines two new types to quickly change implementations
26 typedef Stock StockEntry;
27 typedef std::vector<StockEntry> StockCollection;
28
29 class StockMarket{
30
31 private:
32     StockCollection mStocks;
33     std::string mMarketName;
34
35     void addStock(Stock newStock);
36
37     //*********************************
38     // Method:    findLongestString
39     // FullName:  StockMarket::findLongestString
40     // Access:    private
41     // Returns:   size_t
42     // Qualifier:
43     // Finds the longest string inside stocks
44     //*********************************
45     size_t findLongestString();
46 public:
47     StockMarket(std::string const &marketName);
48     ~StockMarket();
49
50     //*********************************
51     // Method:    getStocks
52     // FullName:  StockMarket::getStocks
53     // Access:    public
54     // Returns:   StockCollection
55     // Qualifier: const
```

```
56      // Returns the stocks inside the stock market
57      //*********************************
58      StockCollection &getStocks();
59
60      //*********************************
61      // Method:    readStocks
62      // FullName:  StockMarket::readStocks
63      // Access:    public
64      // Returns:   void
65      // Qualifier:
66      // Parameter: std::istream & istream
67      // Read stocks from a stream and saves them to stocks
68      //*********************************
69      void readStocks(std::istream &istream);
70
71      //*********************************
72      // Method:    printStockTable
73      // FullName:  StockMarket::printStockTable
74      // Access:    public
75      // Returns:   void
76      // Qualifier:
77      // Parameter: std::ostream & os
78      // Parameter: int sorting
79      // Prints a table with information to the stocks
80      // 1 - sort alphabet
81      // 2 - sort for highest value
82      // 3 - sort for lowest value
83      // 4 - sort for highest diff
84      //*********************************
85      void printStockTable(std::ostream &os, int sorting);
86
87      //*********************************
88      // Method:    Extract
89      // FullName:  StockMarket::Extract
90      // Access:    public
91      // Returns:   StockCollection
92      // Qualifier:
93      // Parameter: StockCollection::iterator begin
94      // Parameter: StockCollection::iterator end
95      // Extracts all values from stocks which are greater than the start value
96      //*********************************
97      StockCollection Extract(StockCollection::iterator begin, StockCollection::
            iterator end);
98
99      //*********************************
100     // Method:    printExtracted
101     // FullName:  StockMarket::printExtracted
102     // Access:    public
103     // Returns:   void
```

```
104     // Qualifier:
105     // Parameter: StockCollection extractedValues
106     // Parameter: std::ostream & os
107     // Prints the result of extract
108     //*************************************
109     void printExtracted(StockCollection extractedValues, std::ostream &os);
110
111     //*************************************
112     // Method:    extractAndPrint
113     // FullName:  StockMarket::extractAndPrint
114     // Access:    public
115     // Returns:   void
116     // Qualifier:
117     // Parameter: std::ostream & os
118     // Extracts and prints with one function call
119     //*************************************
120     void extractAndPrint(std::ostream &os);
121 };
122
123 //*************************************
124 // Method:    simulateStockMarket
125 // FullName:  simulateStockMarket
126 // Access:    public
127 // Returns:   void
128 // Qualifier:
129 // Parameter: StockMarket market
130 // Parameter: size_t numOfDaysToSimulate
131 // Simulates the stock market for the given number of days
132 //*************************************
133 void simulateStockMarket(StockMarket &market, size_t numOfDaysToSimulate);
134
135 #endif
```

Listing A.4: Implementierung der StockMarket Klasse

```
1  ////////////////////////////////////////
2  // Workfile    : StockMarket.cpp
3  // Author      : Matthias Schett
4  // Date        : 27-04-2013
5  // Description : Stock management
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9
10 #include "StockMarket.h"
11 #include "scanner.h"
12 #include "RandomGen.h"
13 #include <exception>
14 #include <iomanip>
15 #include <algorithm>
```

```
16 #include <vector>
17
18 using namespace std;
19
20 // Used to find the longest string for table formatting
21 struct length {
22     bool operator() ( const StockEntry& a, const StockEntry& b ) {
23         return a.getStockName().size() < b.getStockName().size();
24     }
25 };
26
27 struct NameDescending{
28     bool operator() (const StockEntry &a, const StockEntry &b){
29         int compare = strcmp(a.getStockName().c_str(), b.getStockName().c_str
               ());
30
31         if(compare == 0){
32             return false;
33         } else if(compare < 0){
34             return true;
35         } else if(compare > 0){
36             return false;
37         }
38     }
39 };
40
41 struct HighestActualSharePrice{
42     bool operator() (const StockEntry &a, const StockEntry &b){
43         return (a.getActualSharePrice() > b.getActualSharePrice());
44     }
45 };
46
47 struct HighestTotalSharePrice{
48     bool operator() (const StockEntry &a, const StockEntry &b){
49         return (a.getHighestSharePrice() > b.getHighestSharePrice());
50     }
51 };
52
53 struct HighestDiffSharePrice{
54     bool operator() (const StockEntry &a, const StockEntry &b){
55         return ( ( a.getHighestSharePrice() - a.getLowestSharePrice() ) > ( b.
               getHighestSharePrice() - b.getLowestSharePrice() ) );
56     }
57 };
58
59 struct NumOfElements{
60     bool operator() (const StockEntry &a, const StockEntry &b){
61         return (a.getSharePriceCollection().size() > b.getSharePriceCollection
               ().size());
```

```
62      }
63  };
64
65  bool isTbColon(scanner &scan){
66      return scan.symbol_is_colon();
67  }
68
69  bool isTbReal(scanner &scan){
70      return scan.symbol_is_real();
71  }
72
73  bool isTbString(scanner &scan){
74      return scan.symbol_is_string();
75  }
76
77  bool isTbEof(scanner &scan){
78      return scan.symbol_is_eof();
79  }
80
81  string parseStockName(scanner &scan){
82      if(isTbString(scan)){
83          return scan.get_string();
84      }
85      throw std::exception("Unknown format");
86  }
87
88  double parseSharePrice(scanner &scan){
89      if(isTbReal(scan)){
90          return scan.get_real();
91      }
92      throw std::exception("Unknown format");
93  }
94
95  StockMarket::StockMarket(std::string const &marketName) : mMarketName(
        marketName), mStocks(){
96  }
97
98  StockMarket::~StockMarket(){
99  }
100
101 StockCollection &StockMarket::getStocks() {
102     return mStocks;
103 }
104
105 void StockMarket::readStocks( std::istream &istream ){
106
107     scanner scan(istream);
108
109     while(!scan.symbol_is_eof()){
```

```
110
111         if(isTbString(scan)){
112             string stockName = parseStockName(scan);
113             scan.next_symbol();
114             if (!isTbEof(scan)){
115                 if(isTbColon(scan)){
116                     scan.next_symbol();
117                     if (!isTbEof(scan)){
118                         if(isTbReal(scan)){
119                             double actualPrice = parseSharePrice(scan);
120                             scan.next_symbol();
121                             StockEntry newStock(stockName, actualPrice);
122                             addStock(newStock);
123                         }
124                     }
125                 }
126             }
127         } else {
128             throw std::exception("Unknown format");
129         }
130     }
131
132 }
133
134 void StockMarket::addStock(StockEntry newStock){
135     mStocks.push_back(newStock);
136 }
137
138 ostream& hr(std::ostream& os) {
139     return os << "
        ------------------------------------------------------------------------";
140
141 }
142
143 ostream& colSpace(std::ostream& os) {
144     return os << setw(colSpacing) << " ";
145 }
146
147 ostream& colFormatStockValues(std::ostream& os) {
148     return os << setw(colWidthNum) << right << setiosflags(ios::fixed) <<
        setprecision(precision);
149 }
150
151 size_t StockMarket::findLongestString(){
152     StockCollection::iterator it = max_element(getStocks().begin(), getStocks
        ().end(), length());
153
154     return it->getStockName().length();
155 }
```

```
156
157 void printTableHeader(std::ostream &os, size_t longestString){
158     os << hr << endl;
159     os << left << setw(longestString) << "Aktien" <<  colFormatStockValues <<
        "(+/-) Proz" <<  colFormatStockValues << "Aktuell";
160     os <<  colFormatStockValues << "Vortag" <<  colFormatStockValues << "Hoch"
         <<  colFormatStockValues << "Tief";
161     os << endl;
162     os << hr << endl;
163 }
164
165 void printDataLine(StockEntry const &stock, size_t stringLength, std::ostream
    &os){
166     os << left << setw(stringLength) << stock.getStockName() <<
        colFormatStockValues << stock.getStockChangeRate();
167     os <<  colFormatStockValues << stock.getActualSharePrice() <<
        colFormatStockValues << stock.getDayBeforeSharePrice();
168     os <<  colFormatStockValues << stock.getHighestSharePrice() <<
        colFormatStockValues << stock.getLowestSharePrice() << endl;
169 }
170
171 void StockMarket::printStockTable( std::ostream &os, int sorting){
172     size_t longestString = findLongestString();
173     printTableHeader(os, longestString);
174
175     switch(sorting){
176     case 1:
177         sort(getStocks().begin(), getStocks().end(), NameDescending());
178         break;
179     case 2:
180         sort(getStocks().begin(), getStocks().end(), HighestActualSharePrice()
            );
181         break;
182     case 3:
183         sort(getStocks().begin(), getStocks().end(), HighestTotalSharePrice())
            ;
184         break;
185     case 4:
186         sort(getStocks().begin(), getStocks().end(), HighestDiffSharePrice());
187         break;
188     }
189
190
191
192     for(StockCollection::iterator it = getStocks().begin(); it < getStocks().
        end(); ++it){
193         printDataLine(*it, longestString, os);
194     }
195 }
```

```
196
197 StockCollection StockMarket::Extract( StockCollection::iterator begin,
        StockCollection::iterator end ) {
198     StockCollection newCollection;
199     vector<double> values;
200     for(StockCollection::iterator it = begin; it < end; ++it){
201         double startValue = it->getSharePriceCollection().front();
202         vector<double> valuesIterator = it->getSharePriceCollection();
203         for(vector<double>::iterator it2 =  valuesIterator.begin(); it2 <
                valuesIterator.end(); ++it2 ){
204             if(startValue < *it2){
205                 values.push_back(*it2);
206             }
207         }
208         if(!values.empty()){
209             StockEntry entry(it->getStockName(), values);
210             newCollection.push_back(entry);
211         }
212         values.clear();
213     }
214
215     return newCollection;
216 }
217
218 void StockMarket::printExtracted( StockCollection extractedValues, std::
        ostream &os ) {
219
220     sort(extractedValues.begin(), extractedValues.end(), NumOfElements());
221
222     for(StockCollection::iterator it = extractedValues.begin(); it <
            extractedValues.end(); ++it){
223         os << it->getStockName() << ":\t" << it->getSharePriceCollection().
                size();
224         os << endl;
225     }
226
227 }
228
229 void StockMarket::extractAndPrint( std::ostream &os ) {
230     StockCollection col = Extract(getStocks().begin(), getStocks().end());
231
232     printExtracted(col, os);
233 }
234
235 void simulateStock(StockEntry &stock, size_t numOfDaysToSimulate){
236     // With these values the change rate is between -5 and +5 percent
237     int changeRateMax = 500;
238     int changeRateMin = -500;
239     double changeRateDividend = 10000;
```

```
240
241     for(size_t day = 0; day < numOfDaysToSimulate; day++){
242         double changeRate = (double)rgen::GetRandVal(changeRateMin,
                changeRateMax) / changeRateDividend;
243         stock.calcNewValuesOnChangeRate(changeRate);
244     }
245
246 }
247
248 void simulateStockMarket( StockMarket &market, size_t numOfDaysToSimulate ){
249     rgen::Init();
250     for(size_t i = 0; i < market.getStocks().size(); i++){
251         simulateStock(market.getStocks().at(i), numOfDaysToSimulate);
252     }
253 }
```

Listing A.5: Testtreiber

```
1  ////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 27-04-2013
5  // Description : Stock management
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include <vld.h>
11 #include <iostream>
12 #include <fstream>
13 #include "StockMarket.h"
14 #include <exception>
15
16 using namespace std;
17
18 int main(){
19
20     size_t const numOfDaysToSimulate = 365;
21
22     ofstream oFile("OutputA1.txt");
23     ofstream oFile2("OutputA2.txt");
24     try{
25         StockMarket market("Wiener Boerse");
26
27         ifstream file("Input.txt");
28         market.readStocks(file);
29
30         simulateStockMarket(market, numOfDaysToSimulate);
31         oFile << "Table name sorted" << endl;
32         market.printStockTable(oFile, 1);
```

```
33          oFile << endl << endl << endl << "Table highest value sorted" << endl;
34          market.printStockTable(oFile, 2);
35          oFile << endl << endl << endl << "Table lowest value sorted" << endl;
36          market.printStockTable(oFile, 3);
37          oFile << endl << endl << endl << "Table highest diff value sorted" <<
                endl;
38          market.printStockTable(oFile, 4);
39
40          market.extractAndPrint(oFile2);
41
42          file.close();
43      } catch(exception e){
44          oFile << e.what();
45      }
46      oFile.close();
47      oFile2.close();
48
49      cin.get();
50      return 0;
51  }
```