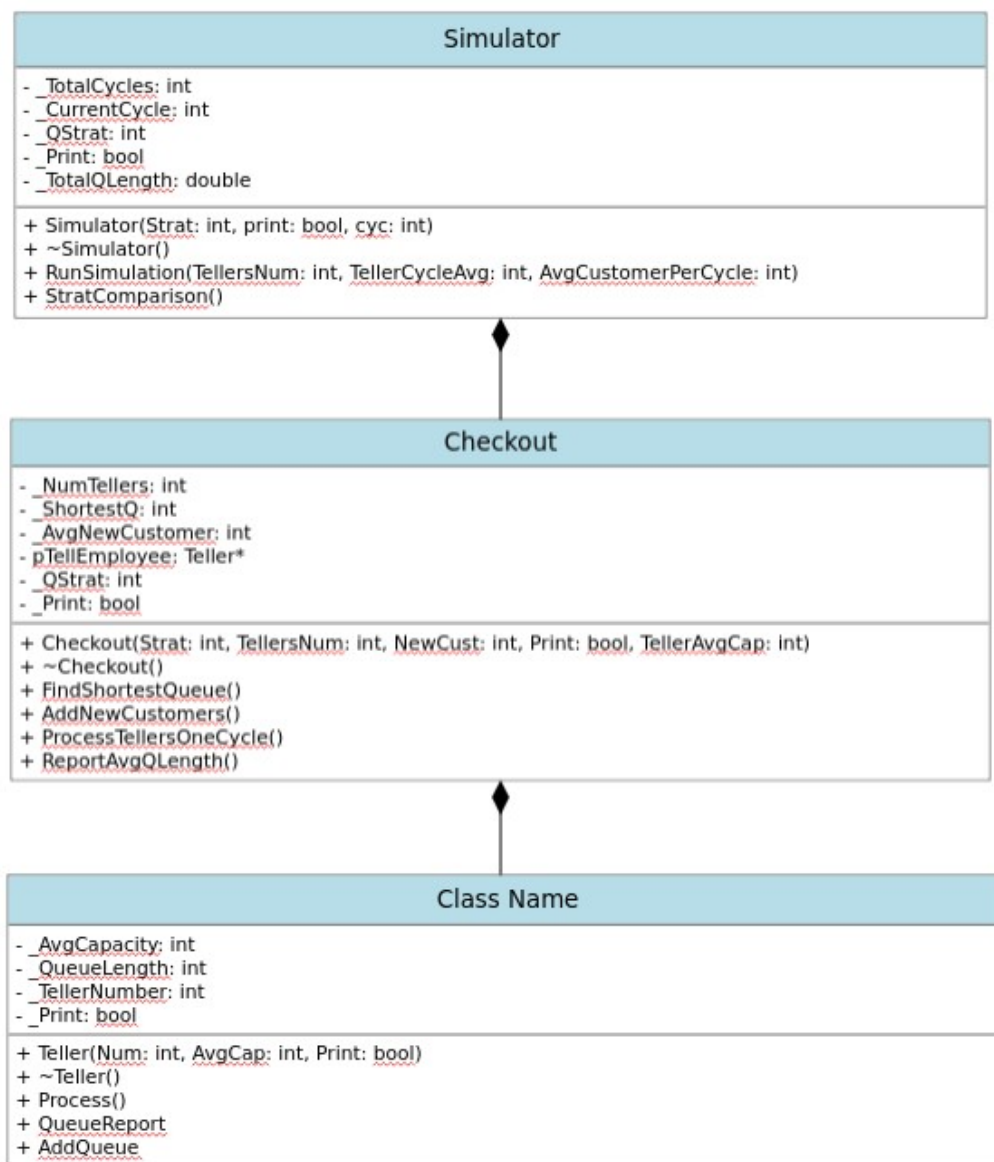# MTRX3760 – Lab 2 Report

SID: 500490778
Tutorial: Lab 3, Thursday 3pm

Functionality:

Shortest Queue Strat Average Queue: 1.17882

Random Queue Strat Average Queue: 2.52481

Design: UML Diagram

```cpp
  1: //SID: 500490778
  2: //Date: 8/22
  3:
  4: #include <iostream>      // std::cout
  5: #include <algorithm>     // std::max
  6: #include <cstdlib>
  7:
  8: #include "queuesim.h"
  9:
 10: //----------Teller Methods--------------
 11: Teller::Teller(int Num, int AvgCap, bool Print)
 12:     :_AvgCapacity(AvgCap),
 13:     _TellerNumber(Num),
 14:     _QueueLength(0),
 15:     _Print(Print)
 16: {
 17:     if(_Print) {
 18:         std::cout << "CTOR Teller, ID: " << _TellerNumber << std::endl;
 19:     }
 20:
 21: }
 22:
 23: //Dtor deletes alllll
 24: Teller::~Teller() {
 25:     if(_Print) {
 26:         std::cout << "DTOR Teller, ID: " << _TellerNumber << std::endl;
 27:     }
 28: }
 29:
 30:
 31: //processes customers one cycle from the queue
 32: void Teller::Process() {
 33:
 34:     if(_Print){
 35:         std::cout << "Teller " << _TellerNumber << " processing with queue length " << _QueueLength;
 36:     }
 37:
 38:     //uniformly distrubuted number averaging avg_capcity taken away from queue length, equals 0 if queue ne
gative
 39:     _QueueLength = std::max(0, _QueueLength - rand()%(_AvgCapacity*2 + 1));
 40:
 41:     if(_Print){
 42:         std::cout << ", reduced after processing to " <<  _QueueLength << std::endl;
 43:     }
 44: };
 45:
 46: //returns queue length
 47: int Teller::QueueReport() {
 48:     return _QueueLength;
 49: }
 50:
 51: //adds to queue
 52: void Teller::AddQueue() {
 53:     _QueueLength++;
 54:
 55:     //Joining message
 56:     if(_Print) {
 57:         std::cout << "Joining a new customer to queue " << _TellerNumber << std::endl;
 58:     }
 59: };
 60:
 61:
 62:
 63: //-------------------Checkout Methods-----------------------
 64:
 65: Checkout::Checkout(int Strat, int TellersNum, int NewCust, bool Print, int TellerAvgCap)
 66:     :_NumTellers(TellersNum),
 67:     _AvgNewCustomer(NewCust),
 68:     _ShortestQ(-1),
 69:     _QStrat(Strat),
 70:     _Print(Print)
 71: {
 72:     if(_Print) {
 73:         std::cout << "CTOR Checkout" << std::endl;
 74:     }
 75:
 76:
 77:     pTellEmployee = new Teller[_NumTellers];
 78:
 79:     for(int i=0; i<_NumTellers; i++) {
 80:
 81:         pTellEmployee[i] = Teller(i, TellerAvgCap, _Print);
 82:
```

```cpp
 83:             //std::cout << i << std::endl;
 84:         }
 85:
 86:  };
 87:
 88:  Checkout::~Checkout() {
 89:         delete [] pTellEmployee;
 90:
 91:
 92:         if(_Print) {
 93:             std::cout << "DTOR Checkout" << std::endl;
 94:         }
 95:  };
 96:
 97:  void Checkout::FindShortestQueue() {
 98:
 99:         //std::cout << "Checkout::FindShortestQueue" << std::endl;
100:
101:         _ShortestQ = 0;
102:         int Length = pTellEmployee[0].QueueReport(); //find length of queue 0
103:
104:
105:         //loop through all tellers to find shortest queue, if tie lowest number teller is allocated
106:         for(int i=1; i<_NumTellers; i++) {
107:             //find if current queue (i) is shorter than shortest queue
108:             if(Length > pTellEmployee[i].QueueReport()) {
109:                 //assign new shortest queue
110:                 _ShortestQ = i;
111:                 Length = pTellEmployee[i].QueueReport();
112:             }
113:         }
114:  };
115:
116:  //adds new customer to existing queues, uses find shortest queue to efficiently disperse new customers
117:  void Checkout::AddNewCustomers() {
118:
119:         int NewCust = rand() % (_AvgNewCustomer*2 + 1);
120:
121:
122:         //display allocation customer message
123:         if(_Print) {
124:             std::cout << "Allocating " << NewCust << " new customers" << std::endl;
125:         }
126:
127:
128:         //find shortest queue and add customer to said queue
129:         for(int i=0; i<NewCust; i++) {
130:
131:             //queue strategy
132:             if(_QStrat == 0) { //allocate customer to the shortest queue
133:                 FindShortestQueue();
134:                 pTellEmployee[_ShortestQ].AddQueue();
135:             } else if (_QStrat == 1) {
136:                 pTellEmployee[rand() % _NumTellers].AddQueue(); //allocates customer to random queue
137:             }
138:
139:         }
140:
141:
142:  };
143:
144:  //tells all tellers to process customers
145:  void Checkout::ProcessTellersOneCycle() {
146:
147:         //loops through all tellers to process cycles
148:         for(int i=0; i<_NumTellers; i++) {
149:             pTellEmployee[i].Process();
150:         }
151:  };
152:
153:  double Checkout::ReportAvgQLength() {
154:
155:         double Total = 0;
156:
157:         //loops through all tellers
158:         for(int i=0; i<_NumTellers; i++) {
159:             Total += (double)pTellEmployee[i].QueueReport();
160:         }
161:
162:         //std::cout << Total / (double)_NumTellers << std::endl;
163:
164:         return Total / (double)_NumTellers;
165:  };
```

```cpp
166:
167: //--------------------Simulation Methdos-------------
168: Simulator::Simulator(int strat, bool print, int cyc)
169:     :_TotalCycles(cyc),
170:     _CurrentCycle(0),
171:     _QStrat(strat),
172:     _Print(print),
173:     _TotalQLength(0)
174: {
175:     if(_Print) {
176:         std::cout << "CTOR Simulator" << std::endl;
177:     }
178:
179: };
180:
181: Simulator::~Simulator() {
182:     if(_Print) {
183:         std::cout << "DTOR Simulator" << std::endl;
184:     }
185:
186: };
187:
188: void Simulator::RunSimulation(int TellersNum, int TellerCycleAvg, int AvgCustomerPerCycle) {
189:
190:     Checkout Coles(_QStrat, TellersNum, AvgCustomerPerCycle, _Print, TellerCycleAvg);
191:
192:     //loop through each cycle, adding new customers and then processing them
193:     for(_CurrentCycle=0; _CurrentCycle< _TotalCycles; _CurrentCycle++) {
194:
195:         if(_Print){
196:             std::cout << "[RunSimulation] simulation cycle " << _CurrentCycle << std::endl;      //prints si
mulation cycle number
197:         }
198:         //adds 0-MaxCustomers to queue
199:         Coles.AddNewCustomers();
200:
201:         //Process Customers from teller queues
202:         Coles.ProcessTellersOneCycle();
203:
204:         //Find Q length
205:         _TotalQLength += Coles.ReportAvgQLength();
206:     }
207: };
208:
209: void Simulator::StratComparison() {
210:
211:     //Run simulation with shortest Q strat
212:     _QStrat = 0;
213:     _TotalQLength = 0;
214:     _CurrentCycle = 0;
215:     RunSimulation();
216:
217:     std::cout << "Shortest Queue Strat Average Queue: " << _TotalQLength / (double)(_CurrentCycle+1) << std
::endl;
218:
219:     _TotalQLength = 0;
220:     _CurrentCycle = 0;
221:     _QStrat = 1;
222:
223:     RunSimulation();
224:     std::cout << "Random Queue Strat Average Queue: " << _TotalQLength / (double)(_CurrentCycle+1) << std::
endl;
225:
226: };
```

```cpp
 1: // SID: 500490778
 2: // 8/2022
 3: /*
 4: This file stores classes that are used to run a shopping line simulator.
 5: The Simulator class will create an instance of the checkout class, and then
 6: the checkout class will create an multple instances of the teller class. The simulator
 7: class will run the simulation for the inputed number of cycles
 8: Checkout class will is capable of controlling the teller class and will simulate adding new
 9: customers and allocating which line
10: Teller runs the most basic methods that simulate the its own line
11: */
12:
13: #ifndef _QUEUESIM_H
14: #define _QUEUESIM_H
15:
16: //Simulates a queue at a teller
17: class Teller {
18:
19:     public:
20:         //tellers number in checkout, average capacity per cycle,
21:         Teller(int Num=0, int AvgCap = 3, bool Print = false);
22:         ~Teller();
23:
24:         void Process();                // Processes one cycles, will remove _TellerCapacity from queue
25:         int QueueReport();             //Returns length of queue
26:         void AddQueue();               //adds to teller queue
27:
28:     private:
29:         int _AvgCapacity;      // how many customers a teller processes in a cycle
30:         int _QueueLength;          // length of queue
31:         int _TellerNumber;         // number teller is at checkout
32:         bool _Print;               //determines if methods print to terminal
33: };
34:
35: //simulates a checkout, runs multiple teller queues
36: class Checkout {
37:
38:     public:
39:         //defualt strat shortest queue, two tellers, default avg 6 new customers, default not print, teller
process average 3/cycle
40:         Checkout(int Strat = 0, int TellersNum = 2, int NewCust = 6, bool Print = false, int TellerAvgCap =
3);
41:         ~Checkout();
42:
43:         void FindShortestQueue();        // finds the shortest queue from an array of queue lengths
44:         void AddNewCustomers();          // adds all new customers to appropriate queues
45:         void ProcessTellersOneCycle();   // processes all tellers one cycle
46:         double ReportAvgQLength();       // returns average length of all teller queues
47:
48:     private:
49:         int _NumTellers;           // number of tellers in the checkout
50:         int _ShortestQ;            // current shortest queue
51:         int _AvgNewCustomer;       //average numbers of new customers added each cycle
52:         Teller* pTellEmployee;     // pointer to an array of tellers
53:         int _QStrat;               // 0 for shortest Q strat, 1 for random Q, determines how AddNewCustome
rs() behaves
54:         bool _Print;               // determines if class prints to terminal
55: };
56:
57: //Runs simulation of checkout multiple cycles
58: class Simulator {
59:
60:     public:
61:         //simulation cycles, number of tellers, average teller process per cycle, average new custmers each
cycle
62:         Simulator(int Strat = 0, bool print = true, int cyc = 1000);
63:         ~Simulator();
64:
65:         //controls the running of the simulation and takes all inputs required for simulation
66:         void RunSimulation(int TellersNum = 3, int TellerCycleAvg = 3, int AvgCustomerPerCycle = 6);
67:
68:         //Runs RunSimulation() for both strategies, prints average queue length of each
69:         void StratComparison();
70:
71:     private:
72:         int _TotalCycles;          // duration of simulation, in cycles
73:         int _CurrentCycle;         // current cycle of simulation
74:         int _QStrat;               // 0 for shortest Q strat, 1 for random Q
75:         bool _Print;               // if function prints to terminal or not
76:         double _TotalQLength;      // Total Queue Length in simulation (adds all queues to length every cy
cles)
77: };
78:
```

```
79:
80:
81: #endif
```

```cpp
 1: // SID: 500490778
 2: // 8/2022
 3: /*
 4: This program runs a queueing simulator to compare the average Queue length if new customer allocated
 5: to the shortest Queue vs allocated to a random Q. Read queuesim.h for details of all classes used
 6: */
 7:
 8: #include <iostream>
 9: #include <cstdlib>                // rand
10: #include <algorithm>             // std::max
11:
12: #include "queuesim.h"
13:
14: int main() {
15:
16:     srand(time(0)); //random number generator same for both strategies, different every run time
17:
18:
19:     //Runs the simulation
20:     Simulator* Output = new Simulator(0, true, 100); //shortest Q strat, printing = true, 100 cycles
21:     Output->RunSimulation();
22:     delete Output;
23:
24:     //Compares random Q strat to shortest Q strat
25:     Simulator Compare(0, false);
26:     Compare.StratComparison();
27:
28: }
```