

```

1  import java.util.*;
2  /*****
3   * Manages MarketPlace operation.
4   *
5   * @author Shane Stacy
6   * @version 12/6/2017
7   *****/
8  public class MarketPlace
9  {
10     /** the average arrival time */
11     private double avgArrivalTime = 0.0;
12
13     /** the average service time */
14     private double avgServiceTime = 0.0;
15
16     /** the number of cashiers */
17     private int numCashiers;
18
19     /** is the checkout area visible? */
20     private boolean checkoutAreaVisible = false;
21
22     /** the current time */
23     private double currentTime = 0.0;
24
25     /** the total wait time */
26     private double totalWaitTime = 0.0;
27
28     /** the average wait time */
29     private double avgWaitTime = 0.0;
30
31     /** the number of customers */
32     private int numCustomers = 0;
33
34     /** the length of the longest line */
35     private int longestLine = 0;
36
37
38     String results = "";
39     ArrayList<Customer> theLine;
40     Customer[] theCashiers;
41     PriorityQueue<GEvent> theEvents;
42     GVrandom rand;
43     GVdeparture nextEvent;
44
45     /** */
46     private final int openTime = 600;
47
48     /** */
49     private final int closeTime = 1080;
50
51     /*****
52     * Creates the MarketPlace
53     *****/
54     public MarketPlace() {
55         avgArrivalTime = 2.5;
56         avgServiceTime = 6.6;

```

```

57     numCashiers = 3;
58     avgWaitTime = 0.0;
59     checkoutAreaVisible = false;
60     currentTime = openTime;
61     rand = new GVrandom();
62     theCashiers = new Customer[numCashiers];
63
64     results = "";
65
66     theLine = new ArrayList<Customer>();
67     theEvents = new PriorityQueue<GEvent>();
68
69 }
70
71 /*****
72  * Gets the number of cashiers in the store
73  * @return int numCashiers the number of cashiers
74  *****/
75 public int getNumCashiers() {
76     return numCashiers;
77 }
78
79 /*****
80  * Gets the arrival time
81  * @return double avgArrivalTime the average arrival time
82  *****/
83 public double getArrivalTime() {
84     return avgArrivalTime;
85 }
86
87 /*****
88  * Gets the service time
89  * @return double avgServiceTime the average service time
90  *****/
91 public double getServiceTime() {
92     return avgServiceTime;
93 }
94
95 /*****
96  * The number of customers served
97  * @return int the list size
98  *****/
99 public int getNumCustomersServed() {
100     return numCustomers;
101 }
102
103 /*****
104  * Gets the store report
105  * @return results the results of the report
106  *****/
107 public String getReport() {
108     return results;
109 }
110
111 /*****
112  * Gets the length of the longest line

```

```

113     * @return the length of the longest line
114     *****/
115     public int getLongestLineLength() {
116         return longestLine;
117     }
118
119     /*****
120     * Gets the average wait time
121     * @return avgWaitTime the average wait time
122     *****/
123     public double getAverageWaitTime() {
124         double avgWaitTime = totalWaitTime / numCustomers;
125         return avgWaitTime;
126     }
127
128     /*****
129     * Initialize some variables
130     * @param int num the number of cashiers
131     * @param double s the average service time
132     * @param double a the average arrival time
133     * @param boolean ck is the checkout area visible
134     *****/
135     public void setParameters(int num, double s, double a, boolean ck) {
136
137         numCashiers = num;
138         avgServiceTime = s;
139         avgArrivalTime = a;
140         checkoutAreaVisible = ck;
141     }
142
143     /*
144     * End of Part 1
145     * Start of Part 2
146     *
147     */
148     /*****
149     * A customer gets in line
150     *****/
151     public void customerGetsInLine() {
152         Customer person = new Customer(currentTime);
153         theLine.add(person);
154         // if the line reaches a new daily high set longestLine equal to it
155         if (theLine.size() > longestLine) {
156             longestLine = theLine.size();
157         }
158         int i = cashierAvailable();
159         // if there is a cashier available, move the next customer to it
160         if (i != -1 && theLine.size() > 0) {
161             customerToCashier(cashierAvailable());
162         }
163         // keep customers coming in if the store is open
164         if (currentTime < closeTime) {
165             double futureTime = randomFutureTime(avgArrivalTime);
166             GVarrival arrive = new GVarrival(this, futureTime);
167             theEvents.add(arrive);
168

```

```

169     }
170
171
172 }
173
174 /*****
175  * A customer pays
176  * @param int num the cashier number
177  *****/
178 public void customerPays (int num) {
179     // if someone is in line move them to the next available cashier
180     if (theLine.size() > 0) {
181         customerToCashier(num);
182     }
183     else {
184         theCashiers[num] = null; // otherwise the cashier is idle
185     }
186
187
188 }
189
190 /*****
191  * Reset the variables
192  *****/
193 public void reset() {
194     currentTime = 600;
195     totalWaitTime = 0.0;
196     avgWaitTime = 0.0;
197     numCustomers = 0;
198     theLine = new ArrayList<Customer>();
199     theEvents = new PriorityQueue<GEvent>();
200
201 }
202
203 /*****
204  * Return the index of the first available cashier
205  * @return int the index of the first available cashier
206  *****/
207 private int cashierAvailable() {
208     int emptyCashier = 0;
209     //look for an open cashier
210     for(int i = 0; i < theCashiers.length; i++) {
211         if (theCashiers[i] == null) {
212             emptyCashier = i;
213             return emptyCashier;
214         }
215     }
216
217     return -1;
218 }
219
220
221 /*****
222  * Returns a future time
223  * @param double avg random number generated by GVrandomnextPosition
224  * @return double the future time

```

```

225     *****/
226 private double randomFutureTime (double avg) {
227     double futureTime = 0.0;
228
229     futureTime = currentTime + rand.nextPoisson(avg);
230     return futureTime;
231 }
232
233 /*****/
234 * Moves customer at front of line to available cashier
235 * @param int num the next available cashier
236 *****/
237 private void customerToCashier (int num) {
238     double futureTime = 0.0;
239     Customer c = theLine.remove(0);
240     theCashiers[num] = c;
241     numCustomers++;
242     totalWaitTime = totalWaitTime + (currentTime - c.getArrivalTime());
243     futureTime = randomFutureTime(avgServiceTime);
244     nextEvent = new GVdeparture(this, futureTime, num);
245     theEvents.add(nextEvent);
246
247 }
248
249 /*****/
250 * Primary method to control simulation from beginning to end
251 *****/
252 public void startSimulation() {
253     reset();
254     GVarrival first = new GVarrival(this, currentTime);
255     theEvents.add(first);
256     // run the store until nothing is left to do
257     while(!theEvents.isEmpty()) {
258         GEvent e = theEvents.poll();
259         currentTime = e.getTime();
260         e.process();
261         if (checkoutAreaVisible == true) {
262             showCheckoutArea();
263         }
264         System.out.println(currentTime);
265     }
266
267     createReport();
268
269 }
270
271 /*****/
272 * Creates the customer
273 * @param double time the customer's arrival time
274 *****/
275 private void showCheckoutArea() {
276     String cashierString = "";
277     String customerString = "";
278     //look for open cashiers and put a C
279     for (int i = 0; i < theCashiers.length; i++) {
280         if (theCashiers[i] == null) {

```

```

281         cashierString = cashierString + "C";
282     }
283     else {
284         cashierString = cashierString + "-"; // otherwise put a -
285     }
286
287 }
288 // put a * to represent the line
289 for (int b = 0; b < theLine.size(); b++) {
290     customerString = customerString + "*";
291 }
292 results = results + formatTime(currentTime) + " " + cashierString + " " + customerString
+ "\n";
293
294 }
295
296 /*****
297  * Creates the report
298  *****/
299 private void createReport() {
300     avgWaitTime = totalWaitTime / numCustomers;
301     results += "Simulation Parameters";
302     results += "\n Number of cashiers: " + numCashiers;
303     results += "\n Average arrival: " + avgArrivalTime;
304     results += "\n Average service: " + avgServiceTime;
305     results += "\n \n Results";
306     results += "\n" + "Average wait time: " + getAverageWaitTime() + " mins";
307     results += "\n" + "Max Line Length: " + getLongestLineLength() + " at " +
formatTime(currentTime);
308     results += "\n" + "Customers served: " + getNumCustomersServed();
309     results += "\n" + "Last departure: " + formatTime(currentTime);
310
311 }
312
313 /*****
314  * Formats a time given the time in minutes
315  * @param double mins time in minutes
316  * @return String a time within a String
317  *****/
318 public String formatTime (double mins) {
319     String formattedTime = "";
320     String aMPM = "";
321     String minutesString;
322     String hoursString;
323     int hours = (int) Math.round(mins / 60);
324     int minutes = (int) Math.round(mins % 60);
325
326     // if time is less than 60 minutes the hour is 12
327     if (mins < 60) {
328         hours = 12;
329
330     }
331     // if time is greater than or equal to 720 minutes the time must be pm
332     if (mins >= 720) {
333         aMPM = "pm";
334     }

```

```
335     }
336     else {
337         aMPM = "am"; // otherwise it's am
338     }
339     // if time is greater or equal to 780 minutes, the hour starts again from 1
340     if (mins >= 780) {
341         hours = hours - 12;
342     }
343
344     minutesString = minutes + "";
345     hoursString = hours + "";
346     // if minutes amount is less than 10 put a zero out front
347     if (minutes < 10) {
348         minutesString = "0" + minutes;
349     }
350     formattedTime = hoursString + ":" + minutesString + " " + aMPM;
351
352     return formattedTime;
353 }
354 }
355
356
357
358
359
```