```java
/**
 * A GVSU Marketplace simulator.
 * <br/>
 * Runs the checkout area to simulate customers waiting in line and
 * being served by a cashier.  This tracks the wait times and line lengths
 * for statistics.  The times at which customers arrive and cashier service
 * times are generated pseudo-randomly around an average time set by the
 * simulation parameters.
 * <br/>
 * The simulation is run by the {@link MarketPlace#startSimulation()} method.
 * That will only output to the console.
 *
 * @author Silas Agnew
 * @version December 4, 2017
 */

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.PriorityQueue;
import java.util.SimpleTimeZone;

public class MarketPlace
{
    private final int OPEN_TIME  = 600;
    private final int CLOSE_TIME = 1080;

    private double                averageArrivalInterval = 0;
    private double                averageServicetime     = 0;
    private int                   numCashiers            = 0;
    private boolean               displayCheckout        = false;
    private double                currentTime            = 0;
    private ArrayList<Customer>   customerList           = null;
    private Customer[]            cashiers               = null;
    private PriorityQueue<GVevent> eventQueue            = null;
    private GVrandom              rand                   = null;
    private String                report                 = "";
    private int                   customersServed        = 0;
    private int                   longestLineLength      = -1;
    private double                lengthTimestamp        = -1;

    // this serves as an accumulative total of wait times then
    // when the simulation finishes it will become the average.
    private double averageWaitTime = 0;

    // More

    /**
     * Constructs a simulated market place.
     */
    public MarketPlace()
    {
        this.averageArrivalInterval = 2.5;
        this.averageServicetime = 6.6;
```

```
 57          this.numCashiers = 3;
 58          this.displayCheckout = false;
 59          this.currentTime = -1;
 60          this.customerList = new ArrayList<>();
 61          this.eventQueue = new PriorityQueue<>();
 62          this.rand = new GVrandom();
 63      }
 64
 65      //-Accessors-----------------------------------------------------//
 66
 67      public int getNumCashiers()
 68      {
 69          return numCashiers;
 70      }
 71
 72      public double getArrivalTime()
 73      {
 74          return averageArrivalInterval;
 75      }
 76
 77      public double getServiceTime()
 78      {
 79          return averageServicetime;
 80      }
 81
 82      public int getNumCustomersServed()
 83      {
 84          return customersServed;
 85      }
 86
 87      public String getReport()
 88      {
 89          return report;
 90      }
 91
 92      public int getLongestLineLength()
 93      {
 94          return longestLineLength;
 95      }
 96
 97      public double getAverageWaitTime()
 98      {
 99          return averageWaitTime;
100      }
101
102      //-Mutators-----------------------------------------------------//
103
104      /**
105       * Sets the parameters for the simulation.
106       *
107       * @param cashiers       Number of cashiers
108       * @param avgServiceTime Average time a customer is with a cashier
109       * @param avgArrival     Average time between customer arrivals
110       * @param displayCheck   Display the checkout information or not
111       */
112      public void setParameters(int cashiers, double avgServiceTime,
```

```
113                                 double avgArrival, boolean displayCheck)
114     {
115         numCashiers = cashiers;
116         averageServicetime = avgServiceTime;
117         averageArrivalInterval = avgArrival;
118         displayCheckout = displayCheck;
119     }
120 
121     /**
122      * Simulates a customer getting in line to checkout. This will add a new
123      * customer to the queue, transfer customers in line to cashiers, if
124      * possible, and push a new event to the event queue.
125      */
126     public void customerGetsInLine()
127     {
128         customerList.add(new Customer(currentTime));
129 
130         // Move customers to cashiers if possible
131         while (cashierAvailable() > -1 && customerList.size() > 0)
132         {
133             customerToCashier(cashierAvailable());
134         }
135 
136         // update stats of longest line and time
137         if (customerList.size() > longestLineLength)
138         {
139             longestLineLength = customerList.size();
140             lengthTimestamp = currentTime;
141         }
142 
143         // Set another customer to arrive
144         // I do not try to compensate the future arrival time for current time
145         // vs closing time because the time they get in line doesn't control the
146         // the time they walked into the store.
147         if (currentTime < CLOSE_TIME)
148         {
149             eventQueue.add(new GVarrival(
150                     this, randomFutureTime(averageArrivalInterval)));
151         }
152     }
153 
154     /**
155      * Simulates a customer paying and leaving the store. Frees the current
156      * cashier and, if there is anymore customers enqueued, assign them to a
157      * cashier.
158      *
159      * @param num Cashier to free from a customer
160      */
161     public void customerPays(int num)
162     {
163         cashiers[num] = null;
164         customersServed++;
165 
166         // Move customers to cashiers if possible
167         while (cashierAvailable() > -1 && customerList.size() > 0)
168         {
```

```
169             customerToCashier(cashierAvailable());
170         }
171     }
172 
173     /**
174      * Runs the simulation.
175      */
176     public void startSimulation()
177     {
178         reset();
179         eventQueue.add(new GVarrival(this, currentTime));
180 
181         while (!eventQueue.isEmpty())
182         {
183             GVevent e = eventQueue.poll();
184             currentTime = e.getTime();
185             e.process();
186 
187             if (displayCheckout) showCheckoutArea();
188         }
189         createReport();
190     }
191 
192     /**
193      * Formats minutes of time with format hh:mm (am/pm).
194      *
195      * @param mins time in minutes
196      * @return A formatted string equivalent to {@code mins}.
197      */
198     public String formatTime(double mins)
199     {
200         // Yes this will be time since unix epoch, but date is not needed
201         SimpleDateFormat fmt = new SimpleDateFormat("h:mma");
202         Date t = new Date((int) mins * 60000); // Convert to ms
203 
204         // Make a timezone because it automatically converts it to local time of
205         // the system
206         fmt.setTimeZone(new SimpleTimeZone(
207                 0, "GVSU MarketPlace Time (GMT)")); // Hi-fives self
208 
209         // Make am/pm lowercase as per the specs
210         return fmt.format(t).replace("AM", "am")
211                 .replace("PM", "pm"); // Hi-fives self again
212     }
213 
214     //-Private Helpers-----------------------------------------------//
215 
216     /**
217      * Resets the simulation.
218      * Does not reset any parameters previously set.
219      */
220     private void reset()
221     {
222         customerList = new ArrayList<>();
223         eventQueue = new PriorityQueue<>();
224         this.cashiers = new Customer[numCashiers];
```

```
225
226         currentTime = OPEN_TIME; // IDK if this is right
227         report = "";
228         customersServed = 0;
229         averageWaitTime = -1;
230         longestLineLength = -1;
231         lengthTimestamp = -1;
232     }
233
234     /**
235      * @return The first available cashier's index. If there is none, -1.
236      */
237     private int cashierAvailable()
238     {
239         for (int i = 0; i < cashiers.length; i++)
240         {
241             if (cashiers[i] == null) return i;
242         }
243         return -1;
244     }
245
246     /**
247      * Generates a random time based on {@code avg} and adds it to the current
248      * time to ensure that it is in the future of the simulation.
249      *
250      * @param avg Number to base generation on.
251      * @return A random time in the future of the simulation.
252      */
253     private double randomFutureTime(double avg)
254     {
255         return currentTime + rand.nextPoisson(avg);
256     }
257
258     /**
259      * Moves a customer from the line to a cashier of index {@code num}.
260      *
261      * @param num Index of the receiving cashier.
262      */
263     private void customerToCashier(int num)
264     {
265         cashiers[num] = customerList.remove(0);
266         double waitTime = currentTime - cashiers[num].getArrivalTime();
267
268         averageWaitTime += (waitTime < 0) ? 0 : waitTime;
269         eventQueue.add(new GVdeparture(
270                 this, randomFutureTime(averageServicetime), num));
271     }
272
273     /**
274      * Appends a timestamp and checkout information to the simulation report.
275      */
276     private void showCheckoutArea()
277     {
278         // Timestamp
279         report += formatTime(currentTime) + " ";
280
```

```
281         // Cashiers
282         for (int i = 0; i < this.cashiers.length; i++)
283         {
284             if (this.cashiers[i] == null) report += "_";
285             else report += "C";
286         }
287 
288         report += " ";
289 
290         // Queue
291         for (int i = 0; i < customerList.size(); i++)
292             report += "*";
293         report += "\n";
294     }
295 
296     /**
297      * Compiles simulation data and stats and appends them to the simulation report.
298      */
299     private void createReport()
300     {
301         // Sim params
302         report += "\n\nSIMULATION PARAMETERS\nNumber of cashiers: " + cashiers.length
303                 + "\nAverage arrival: " + averageArrivalInterval
304                 + "\nAverage service: " + averageServicetime;
305 
306         // Calculate average
307         averageWaitTime /= customersServed;
308 
309         NumberFormat fmt = new DecimalFormat("#0.00");
310 
311         // Results
312         report += "\n\nRESULTS\nAverage wait time: " + fmt.format(averageWaitTime)
313                 + " mins" + "\nMax line length: " + longestLineLength + " at "
314                 + formatTime(lengthTimestamp) + "\nCustomers served: "
315                 + customersServed + "\nLast departure: " + formatTime(currentTime);
316     }
317 }
318 
```