```java
//******************************************************************************
//
// File:    Generator.java
// Package: ---
// Unit:    Class Generator
//
//******************************************************************************

import edu.rit.numeric.ExponentialPrng;
import edu.rit.numeric.ListSeries;
import edu.rit.numeric.Series;
import edu.rit.sim.Event;
import edu.rit.sim.Simulation;
import edu.rit.util.Random;

/**
 * Class Generator generates requests for the web server simulations.
 *
 * @author  Alan Kaminsky
 * @author Jimi Ford (jhf3617)
 * @version 5-2-2015
 */
public class Generator
{
    // private data members

    private Simulation sim;
    private ExponentialPrng tpktPrng;
    private Random prng;
    private final int npkt;
    private Routable source;
    private Link link;
    private ListSeries respTimeSeries;
    private ListSeries respTimeLargePackets;
    private ListSeries respTimeSmallPackets;
    private int largePackets;
    private int smallPackets;

    /**
     * Create a new request generator.
     *
     * @param  sim     Simulation.
     * @param  rpkt    Packet generation mean rate.
     * @param  npkt    Number of packets.
     * @param  prng    Pseudorandom number generator.
     * @param  source  First host in network sending the packets.
     */
    public Generator (Simulation sim, double rpkt, int npkt, Random prng,
            Routable source, Link link) {
        this.sim = sim;
        this.tpktPrng = new ExponentialPrng (prng, rpkt);
        this.npkt = npkt;
        this.source = source;
        this.prng = prng;
        respTimeSeries = new ListSeries();
        respTimeLargePackets = new ListSeries();
        respTimeSmallPackets = new ListSeries();
        largePackets = 0;
```

```
59          smallPackets = 0;
60          this.link = link;
61          generatePacket();
62      }
63
64      /**
65       * Generate the next packet.
66       */
67      private void generatePacket() {
68          Packet p = new Packet (prng, sim, respTimeSeries, respTimeLargePackets,
69                  respTimeSmallPackets);
70          if(link.ready()) {
71              source.startSending (p, link);
72          }
73          if(p.isLarge) ++largePackets;
74          else ++smallPackets;
75          if (totalPackets() < npkt) {
76              sim.doAfter (tpktPrng.next(), new Event() {
77                  public void perform() {
78                      generatePacket();
79                  }
80              });
81          }
82      }
83
84      /**
85       * Returns a data series containing the response time statistics of the
86       * generated requests.
87       *
88       * @return  Response time series.
89       */
90      public Series responseTimeSeries() {
91          return respTimeSeries;
92      }
93
94      /**
95       * Returns a data series containing the response time statistics of the
96       * larger packets
97       *
98       * @return  Response time series.
99       */
100     public Series responseTimeLarge() {
101         return respTimeLargePackets;
102     }
103
104     /**
105      * Returns a data series containing the response time statistics of the
106      * smaller packets.
107      *
108      * @return  Response time series.
109      */
110     public Series responseTimeSmall() {
111         return respTimeSmallPackets;
112     }
113
114     /**
115      * Returns the total number of packets currently generated
116      */
```

```
117     public int totalPackets() {
118         return largePackets + smallPackets;
119     }
120 
121     /**
122      * Returns the response time statistics of the generated requests.
123      *
124      * @return  Response time statistics (mean, standard deviation, variance).
125      */
126     public Series.Stats responseTimeStats() {
127         return respTimeSeries.stats();
128     }
129 
130     /**
131      * Returns the drop fraction of the generated packets.
132      */
133     public double totalDropFraction() {
134         return (double)(totalPackets() - respTimeSeries.length())
135                 /(double)totalPackets();
136     }
137 
138     /**
139      * Returns the drop fraction of the large packets generated
140      */
141     public double largePacketDropFraction() {
142         return (double)(largePackets - respTimeLargePackets.length())
143                 /(double)largePackets;
144     }
145 
146     /**
147      * Returns the drop fraction of the small packets generated
148      */
149     public double smallPacketDropFraction() {
150         return (double)(smallPackets - respTimeSmallPackets.length())
151                 /(double)smallPackets;
152     }
153 }
```