```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;
        smallPackets = 0;
        this.link = link;
        generatePacket();
    }

    /**
     * Generate the next packet.
     */
    private void generatePacket() {
        Packet p = new Packet (prng, sim, respTimeSeries, respTimeLargePackets,
                respTimeSmallPackets);
        if(link.ready()) {
            source.startSending (p, link);
        }
        if(p.isLarge) ++largePackets;
        else ++smallPackets;
        if (totalPackets() < npkt) {
            sim.doAfter (tpktPrng.next(), new Event() {
                public void perform() {
                    generatePacket();
                }
            });
        }
    }

    /**
     * Returns a data series containing the response time statistics of the
     * generated requests.
     *
     * @return  Response time series.
     */
    public Series responseTimeSeries() {
        return respTimeSeries;
    }

    /**
     * Returns a data series containing the response time statistics of the
     * larger packets
     *
     * @return  Response time series.
     */
    public Series responseTimeLarge() {
        return respTimeLargePackets;
    }

    /**
     * Returns a data series containing the response time statistics of the
     * smaller packets.
     *
     * @return  Response time series.
     */
    public Series responseTimeSmall() {
        return respTimeSmallPackets;
    }

    /**
     * Returns the total number of packets currently generated
     */
    public int totalPackets() {
        return largePackets + smallPackets;
    }

    /**
     * Returns the response time statistics of the generated requests.
     *
     * @return  Response time statistics (mean, standard deviation, variance).
     */
    public Series.Stats responseTimeStats() {
        return respTimeSeries.stats();
    }

    /**
     * Returns the drop fraction of the generated packets.
     */
    public double totalDropFraction() {
        return (double)(totalPackets() - respTimeSeries.length())
                /(double)totalPackets();
    }

    /**
     * Returns the drop fraction of the large packets generated
     */
    public double largePacketDropFraction() {
        return (double)(largePackets - respTimeLargePackets.length())
                /(double)largePackets;
    }

    /**
     * Returns the drop fraction of the small packets generated
     */
    public double smallPacketDropFraction() {
        return (double)(smallPackets - respTimeSmallPackets.length())
                /(double)smallPackets;
    }
}
```

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

import edu.rit.sim.Simulation;

/**
 * Class Host provides the server in the web simulation. The server's
 * request processing time is exponentially distributed with a given mean.
 * Requests are added to the server's queue at any time. The queue has a given
 * maximum size.
 *
 * @author Jimi Ford (jhf3617)
 * @version 22-Apr-2015
 */
public class Host extends Routable
{

    /**
     * Construct a new server. The server's request processing time is
     * exponentially distributed with the given mean. The server's request queue
     * has the given maximum size.
     *
     * @param  sim    Simulation.
     */
    public Host(Simulation sim) {
        super(sim);
    }

    /**
     * Called when this routable object finished receiving a packet on a certain
     * link
     * @param packet the packet this object received
     * @param link the link that the packet was received on
     */
    public void receivePacket(final Packet packet, final Link link) {
        link.open();
        packet.finish();
    }
}
```

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

import edu.rit.sim.Simulation;

/**
 * Class Link represents a connection between two routable objects. Links are
 * <I>closed</I> if a packet is currently transmitting on them and <I>open</I>
 * if they are ready to be transmitted on.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-6-2015
 */
public class Link {

    /**
     * default bit rate used in this project
     * <P>9600 bits/sec</P>
     */
    public static final int DEFAULT_BIT_RATE = 9600;

    /**
     * true if this link has an infinite bit rate
     */
    public final boolean infiniteBitRate;

    /**
     * the bit rate of this link
     */
    public final double bitRate;

    // private data members

    private final Routable r1;
    private final Routable r2;
    private final Simulation sim;
    private double closeStarted;
    private double closeFinished;
    private double totalTimeSpentClosed;
    private boolean ready;


    /**
     * construct a link with the default finit bit rate between two routables
     *
     * @param sim the simulation reference
     * @param r1 one of the routable objects
     * @param r2 the other routable object
     */
    public Link(Simulation sim, Routable r1, Routable r2) {
        this(sim, false, r1, r2);
    }

    /**
     * construct a link with specified finite or infinite bit rate
     *
     * @param sim the simulation reference
     * @param infiniteBitRate set to true for infinite bit rate, false for
     * default finite bit rate
     * @param r1 one of the routable objects
     * @param r2 the other routable object
     */
    public Link(Simulation sim, boolean infiniteBitRate, Routable r1,
            Routable r2) {
        this.sim = sim;
        this.r1 = r1;
        this.r2 = r2;
        this.ready = true;
        this.infiniteBitRate = infiniteBitRate;
        this.bitRate = infiniteBitRate ? Double.POSITIVE_INFINITY :
            DEFAULT_BIT_RATE;
        this.totalTimeSpentClosed = 0;
    }

    /**
     * get the other routable object attached to this link compared to the
     * current one
     *
     * @param current the current routable object querying for the other
     * attached routable object
     * @return the routable object that is not equal to the current one
     */
    public Routable other(Routable current) {
        return this.r1.equals(current) ? r2 : r1;
    }

    /**
     * get the current state of the link
     *
     * @return true if the link is ready to pass another packet along it; false
     * otherwise
     */
    public boolean ready() {
        return this.ready;
    }

    /**
     * close this link off so that other packets may not be transmitted on it
     * until open() is called
     *
     * @throws IllegalStateException if the link is not ready to be closed and
     * this link has a finite bit-rate
     */
    public void close() throws IllegalStateException {
        if(!this.infiniteBitRate) {
            if(!this.ready) {
                throw new IllegalStateException();
            }
            this.ready = false;
            this.closeStarted = sim.time();
        }
```

```java
    }

    /**
     * open this link so that other packets may be transmitted on it
     */
    public void open() {
        this.ready = true;
        this.closeFinished = sim.time();
        this.totalTimeSpentClosed += (this.closeFinished - this.closeStarted);
    }

    /**
     * Return the amount of time this link was closed as a fraction of the
     * total amount of time in the simulation.
     */
    public double fractionClosed() {
        return this.totalTimeSpentClosed / sim.time();
    }
}

```

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

import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.Series;
import edu.rit.sim.Simulation;
import edu.rit.util.Random;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Class MrPotatoHead is the hot potato simulation main program. It simulates
 * a network in which routers use hot potato routing and uses Prof. Alan
 * Kaminsky's pj2 library to aid in this discrete event simulation.
 *
 * @author  Alan Kaminsky
 * @author Jimi Ford (jhf3617)
 * @version 5-3-2015
 */
public class MrPotatoHead
{
    private static double rlb;
    private static double rub;
    private static double rdelta;
    private static int npkt;
    private static long seed;

    private static Random prng;
    private static Simulation sim;
    private static String prefix;
    private static Generator gen;

    /**
     * Main program to simulate hot-potato routing
     *
     * @param args command line arguments
     */
    public static void main(String[] args)
    {
        // Parse command line arguments.
        if (args.length != 5 && args.length != 6) usage();
        rlb = Double.parseDouble (args[0]);
        rub = Double.parseDouble (args[1]);
        rdelta = Double.parseDouble (args[2]);
        if(rlb <= 0) rlb = rdelta;
        npkt = Integer.parseInt (args[3]);
        seed = Long.parseLong (args[4]);
        prefix = args.length == 6 ? args[5] : "potato";
        // Set up pseudorandom number generator.
        prng = new Random (seed);

        // Set up plot data series.
        ListXYSeries respTimeSeries = new ListXYSeries();
        ListXYSeries respTimeLargeSeries = new ListXYSeries();
        ListXYSeries respTimeSmallSeries = new ListXYSeries();
        ListXYSeries dropFracSeries = new ListXYSeries();
        ListXYSeries dropFracLargeSeries = new ListXYSeries();
        ListXYSeries dropFracSmallSeries = new ListXYSeries();

        ListXYSeries aDrop = new ListXYSeries();
        ListXYSeries bDrop = new ListXYSeries();
        ListXYSeries cDrop = new ListXYSeries();
        ListXYSeries dDrop = new ListXYSeries();

        ListXYSeries aReRoute = new ListXYSeries();
        ListXYSeries bReRoute = new ListXYSeries();
        ListXYSeries cReRoute = new ListXYSeries();
        ListXYSeries dReRoute = new ListXYSeries();

        ListXYSeries adActivity = new ListXYSeries();
        ListXYSeries bdActivity = new ListXYSeries();
        ListXYSeries cdActivity = new ListXYSeries();
        ListXYSeries d2Activity = new ListXYSeries();

        ListXYSeries abActivity = new ListXYSeries();
        ListXYSeries acActivity = new ListXYSeries();
        ListXYSeries baActivity = new ListXYSeries();
        ListXYSeries bcActivity = new ListXYSeries();
        ListXYSeries caActivity = new ListXYSeries();
        ListXYSeries cbActivity = new ListXYSeries();
        ListXYSeries daActivity = new ListXYSeries();
        ListXYSeries dbActivity = new ListXYSeries();
        ListXYSeries dcActivity = new ListXYSeries();

        // Sweep mean request rate.
        System.out.printf ("Mean\tResp\tResp\tResp\tDrop\tDrop\tDrop%n");
        System.out.printf ("Pkt\tTime\tTime\tTime\tFrac\tFrac\tFrac%n");
        System.out.printf ("Rate\tTotal\tLarge\tSmall\tTotal\tLarge\tSmall%n");
        StringBuilder builder = new StringBuilder();
        builder.append(
            String.format("Mean\tResp\tResp\tResp\tDrop\tDrop\tDrop%n"));
        builder.append(
            String.format("Pkt\tTime\tTime\tTime\tFrac\tFrac\tFrac%n"));
        builder.append(
            String.format("Rate\tTotal\tLarge\tSmall\tTotal\tLarge\tSmall%n"));
        double rate;
        for (int i = 0; (rate = rlb + i*rdelta) <= rub; ++ i)
        {
            // Set up simulation.
            sim = new Simulation();
            Host h1, h2;
            Router a, b, c, d;

            h1 = new Host(sim);
            h2 = new Host(sim);
            d = new Router(prng, sim);
            a = new Router(prng, sim);
            b = new Router(prng, sim);
            c = new Router(prng, sim);
            Link
                ab = new Link(sim, a, b),
                ac = new Link(sim, a, c),
                ad = new Link(sim, a, d),
                ba = new Link(sim, b, a),
                bc = new Link(sim, b, c),
                bd = new Link(sim, b, d),
                ca = new Link(sim, c, a),
                cb = new Link(sim, c, b),
                cd = new Link(sim, c, d),
                da = new Link(sim, d, a),
                db = new Link(sim, d, b),
                dc = new Link(sim, d, c),
                d2 = new Link(sim, d, h2);
            // preferred link
            a.setPrimary(ad);
            b.setPrimary(bd);
            c.setPrimary(cd);
            d.setPrimary(d2);
            // secondary links
            a.addSecondary(ab);
            a.addSecondary(ac);
            b.addSecondary(ba);
            b.addSecondary(bc);
            c.addSecondary(ca);
            c.addSecondary(cb);
            d.addSecondary(da);
            d.addSecondary(db);
            d.addSecondary(dc);

            // Set up request generator and generate first request.
            gen = new Generator (sim, rate, npkt, prng, h1,
                    new Link(sim, true, h1, a));

            // Run the simulation.
            sim.run();

            // Print results.
            Series.Stats totalStats = gen.responseTimeStats();
            Series.Stats largeStats = gen.responseTimeLarge().stats();
            Series.Stats smallStats = gen.responseTimeSmall().stats();
            System.out.printf ("%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f%n",
                    rate, totalStats.mean, largeStats.mean, smallStats.mean,
                    gen.totalDropFraction(), gen.largePacketDropFraction(),
                    gen.smallPacketDropFraction());
            builder.append(String.format(
                    "%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f%n",
                    rate, totalStats.mean, largeStats.mean, smallStats.mean,
                    gen.totalDropFraction(), gen.largePacketDropFraction(),
                    gen.smallPacketDropFraction()));
            // Record results for plot.
            respTimeSeries.add (rate, totalStats.mean);
            respTimeLargeSeries.add(rate, largeStats.mean);
            respTimeSmallSeries.add(rate, smallStats.mean);
            dropFracSeries.add (rate, gen.totalDropFraction());
            dropFracLargeSeries.add(rate, gen.largePacketDropFraction());
            dropFracSmallSeries.add(rate, gen.smallPacketDropFraction());
            aDrop.add(rate, a.dropFraction(npkt));
            bDrop.add(rate, b.dropFraction(npkt));
            cDrop.add(rate, c.dropFraction(npkt));
```

```
            dDrop.add(rate, d.dropFraction(npkt));
            aReRoute.add(rate, a.reRouteFraction());
            bReRoute.add(rate, b.reRouteFraction());
            cReRoute.add(rate, c.reRouteFraction());
            dReRoute.add(rate, d.reRouteFraction());
            // primary link activity
            adActivity.add(rate, ad.fractionClosed());
            bdActivity.add(rate, bd.fractionClosed());
            cdActivity.add(rate, cd.fractionClosed());
            d2Activity.add(rate, d2.fractionClosed());
            // secondary link activity
            abActivity.add(rate, ab.fractionClosed());
            acActivity.add(rate, ac.fractionClosed());
            baActivity.add(rate, ba.fractionClosed());
            bcActivity.add(rate, bc.fractionClosed());
            caActivity.add(rate, ca.fractionClosed());
            cbActivity.add(rate, cb.fractionClosed());
            daActivity.add(rate, da.fractionClosed());
            dbActivity.add(rate, db.fractionClosed());
            dcActivity.add(rate, dc.fractionClosed());
        }

        try {
            new PlotHandler(prefix, dropFracSeries, respTimeSeries,
                    dropFracLargeSeries, respTimeLargeSeries,
                    dropFracSmallSeries, respTimeSmallSeries,
                    aDrop, bDrop, cDrop, dDrop,
                    aReRoute, bReRoute, cReRoute, dReRoute,
                    adActivity, bdActivity, cdActivity, d2Activity,
                    abActivity, acActivity, baActivity, bcActivity, caActivity,
                    cbActivity, daActivity, dbActivity, dcActivity).write();
            PrintWriter tableWriter = new PrintWriter(prefix + "-table.tsv");
            tableWriter.print(builder.toString());
            tableWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Print a usage message and exit.
     */
    private static void usage()
    {
        System.err.println ("Usage: java MrPotatoHead <rlb> <rub> <rdelta> "
                + "<npkt> <seed> [<file-prefix> (optional)]");
        System.err.println ("<rlb> = Mean packet rate lower bound");
        System.err.println ("<rub> = Mean packet rate upper bound");
        System.err.println ("<rdelta> = Mean packet rate delta");
        System.err.println ("<npkt> = Number of packets");
        System.err.println ("<seed> = Random seed");
        System.err.println ("<file-prefix> = optional file prefix, "
                + "default = \"potato\"");
        System.exit (1);
    }
}
```

```
//******************************************************************************
//
// File:    Packet.java
// Package: ---
// Unit:    Packet Link
//
//******************************************************************************

import edu.rit.numeric.ListSeries;
import edu.rit.sim.Simulation;
import edu.rit.util.Random;

/**
 * Class Packet provides a packet model in the web simulation. It contains the
 * logic necessary to determine the size of the packet and the amount of time
 * it would take to transmit along a link. It also reports to several instances
 * of ListSeries objects that keep track of the response time of packets based
 * on their size.
 *
 * @author  Alan Kaminsky
 * @author Jimi Ford (jhf3617)
 * @version 5-6-2015
 */
public class Packet
{
    /**
     * size of the packet in bits
     */
    public final int size;

    /**
     * unique identifier across all other packets
     */
    public final int id;

    /**
     * true if this packet is 576 bytes, false otherwise
     */
    public final boolean isLarge;

    // private data member

    private static int idCounter = 0;
    private Simulation sim;
    private double startTime;
    private double finishTime;
    private ListSeries respTimeSeries;
    private ListSeries respTimeLargePackets;
    private ListSeries respTimeSmallPackets;

    private static final int
        SMALL = 40 * Byte.SIZE,
        LARGE = 576 * Byte.SIZE;


    /**
     * Construct a new packet. The packet's response time will be recorded in
     * the ListSeries.
     * @param prng a pseudorandom number generator
     * @param sim the current simulation object
     * @param series the series to keep track of response times in
     * @param seriesLargePackets series to keep track of large packet response
     * times in
     * @param seriesSmallPackets series to keep track of small packet response
     * times in
     */
    public Packet(Random prng, Simulation sim, ListSeries series,
            ListSeries seriesLargePackets, ListSeries seriesSmallPackets) {
        this.id = ++ idCounter;
        this.sim = sim;
        this.startTime = sim.time();
        this.size = prng.nextDouble() < .5 ? SMALL : LARGE;
        this.isLarge = this.size == LARGE;
        this.respTimeSeries = series;
        this.respTimeLargePackets = seriesLargePackets;
        this.respTimeSmallPackets = seriesSmallPackets;
    }

    /**
     * get the time it would take this packet to transmit along a given link
     *
     * @param link the given link to transmit on
     * @return the time in seconds it would take to transmit on the given link
     */
    public double transmitTime(Link link) {
        if(link.infiniteBitRate) {
            return 0;
        }
        return ((double)size) / link.bitRate;
    }

    /**
     * Mark this request as finished. The request's finish time is set to the
     * current simulation time. The request's response time is recorded in the
     * response time series.
     */
    public void finish()
    {
        finishTime = sim.time();
        respTimeSeries.add (responseTime());
        if(isLarge) respTimeLargePackets.add(responseTime());
        else respTimeSmallPackets.add(responseTime());
    }

    /**
     * Returns this request's response time.
     *
     * @return  Response time.
     */
    public double responseTime()
    {
        return finishTime - startTime;
    }

    /**
     * Returns a string version of this request.
     *
     * @return  String version.
     */
    public String toString()
    {
        return "Packet " + id;
    }
}
```

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

import java.awt.BasicStroke;
import java.awt.Color;
import java.io.IOException;
import java.text.DecimalFormat;

import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.plot.Dots;
import edu.rit.numeric.plot.Plot;

/**
 * Class PlotHandler is the delegate for dealing with visualizing the data
 * generated by the "number crunching" program, MrPotatoHead.
 * Its purpose is to be instantiated in MrPotatoHead with the data to plot,
 * where the write() method should then be called.
 * <P>
 * Running this program and specifying in the command line arguments the plot
 * files previously generated will open a graphical representation of these
 * plots for each file.
 * </P>
 * @author Jimi Ford
 * @version 5-6-2015
 *
 */
public class PlotHandler {

    // private data members
    private final String rtTotalFile;
    private final String dfTotalFile;
    private final String rtLargeFile;
    private final String dfLargeFile;
    private final String rtSmallFile;
    private final String dfSmallFile;
    private final String routerDropFile;
    private final String reRouteFile;
    private final String primaryActivityFile;
    private final String secondaryActivityFile;
    private final ListXYSeries dfTotal;
    private final ListXYSeries rtTotal;
    private final ListXYSeries dfLarge;
    private final ListXYSeries rtLarge;
    private final ListXYSeries dfSmall;
    private final ListXYSeries rtSmall;
    private final ListXYSeries aDrop;
    private final ListXYSeries bDrop;
    private final ListXYSeries cDrop;
    private final ListXYSeries dDrop;
    private final ListXYSeries aReRoute;
    private final ListXYSeries bReRoute;
    private final ListXYSeries cReRoute;
    private final ListXYSeries dReRoute;
    private final ListXYSeries adActivity;
    private final ListXYSeries bdActivity;
    private final ListXYSeries cdActivity;
    private final ListXYSeries d2Activity;
    private final ListXYSeries abActivity;
    private final ListXYSeries acActivity;
    private final ListXYSeries baActivity;
    private final ListXYSeries bcActivity;
    private final ListXYSeries caActivity;
    private final ListXYSeries cbActivity;
    private final ListXYSeries daActivity;
    private final ListXYSeries dbActivity;
    private final ListXYSeries dcActivity;


    /**
     * Construct a new PlotHandler object
     *
     * @param prefix the prefix to use for saving the files
     * @param dfTotal the xy-series that contains the drop fraction info
     * @param rtTotal the xy-series that contains the response time info
     * @param dfLarge series containing the drop fraction for large packets
     * @param rtLarge series containing the response time for large packets
     * @param dfSmall series containing the drop fraction for small packets
     * @param rtSmall series containing the response time for small packets
     * @param aDrop series containing the drop fraction of router a
     * @param bDrop series containing the drop fraction of router b
     * @param cDrop series containing the drop fraction of router c
     * @param dDrop series containing the drop fraction of router d
     * @param aReRoute series containing the re-route fraction of router a
     * @param bReRoute series containing the re-route fraction of router b
     * @param cReRoute series containing the re-route fraction of router c
     * @param dReRoute series containing the re-route fraction of router d
     * @param adActivity series containing activity fraction of link ad
     * @param bdActivity series containing activity fraction of link bd
     * @param cdActivity series containing activity fraction of link cd
     * @param d2Activity series containing activity fraction of link d2
     * @param abActivity series containing activity fraction of link ab
     * @param acActivity series containing activity fraction of link ac
     * @param baActivity series containing activity fraction of link ba
     * @param bcActivity series containing activity fraction of link bc
     * @param caActivity series containing activity fraction of link ca
     * @param cbActivity series containing activity fraction of link cb
     * @param daActivity series containing activity fraction of link da
     * @param dbActivity series containing activity fraction of link db
     * @param dcActivity series containing activity fraction of link dc
     */
    public PlotHandler(String prefix,
        ListXYSeries dfTotal, ListXYSeries rtTotal,
        ListXYSeries dfLarge, ListXYSeries rtLarge,
        ListXYSeries dfSmall, ListXYSeries rtSmall,
        ListXYSeries aDrop, ListXYSeries bDrop, ListXYSeries cDrop,
        ListXYSeries dDrop, ListXYSeries aReRoute, ListXYSeries bReRoute,
        ListXYSeries cReRoute, ListXYSeries dReRoute,
        ListXYSeries adActivity, ListXYSeries bdActivity,
        ListXYSeries cdActivity, ListXYSeries d2Activity,
        ListXYSeries abActivity, ListXYSeries acActivity,
        ListXYSeries baActivity, ListXYSeries bcActivity,
        ListXYSeries caActivity, ListXYSeries cbActivity,
        ListXYSeries daActivity, ListXYSeries dbActivity,
        ListXYSeries dcActivity) {
        rtTotalFile = prefix + "-traversal-time.dwg";
        dfTotalFile = prefix + "-drop-fraction.dwg";
        rtLargeFile = prefix + "-traversal-time-large.dwg";
        rtSmallFile = prefix + "-traversal-time-small.dwg";
        dfLargeFile = prefix + "-drop-fraction-large.dwg";
        dfSmallFile = prefix + "-drop-fraction-small.dwg";
        routerDropFile = prefix + "-router-drop-fraction.dwg";
        reRouteFile = prefix + "-re-route-fraction.dwg";
        primaryActivityFile = prefix + "-primary-link-activity-fraction.dwg";
        secondaryActivityFile = prefix +
                "-secondary-link-activity-fraction.dwg";
        this.dfTotal = dfTotal;
        this.rtTotal = rtTotal;
        this.dfLarge = dfLarge;
        this.rtLarge = rtLarge;
        this.dfSmall = dfSmall;
        this.rtSmall = rtSmall;
        this.aDrop = aDrop;
        this.bDrop = bDrop;
        this.cDrop = cDrop;
        this.dDrop = dDrop;
        this.aReRoute = aReRoute;
        this.bReRoute = bReRoute;
        this.cReRoute = cReRoute;
        this.dReRoute = dReRoute;
        this.adActivity = adActivity;
        this.bdActivity = bdActivity;
        this.cdActivity = cdActivity;
        this.d2Activity = d2Activity;
        this.abActivity = abActivity;
        this.acActivity = acActivity;
        this.baActivity = baActivity;
        this.bcActivity = bcActivity;
        this.caActivity = caActivity;
        this.cbActivity = cbActivity;
        this.daActivity = daActivity;
        this.dbActivity = dbActivity;
        this.dcActivity = dcActivity;
    }

    /**
     * Save the plot information into files and display the plots.
     *
     * @throws IOException if it can't write to the file specified
     */
    public void write() throws IOException {
        write("Total", "0.0", dfTotal, dfTotalFile, rtTotal, rtTotalFile);
        write("Large Pkt", "0.0", dfLarge, dfLargeFile, rtLarge, rtLargeFile);
        write("Small Pkt", "0.00", dfSmall, dfSmallFile, rtSmall, rtSmallFile);
        writeRouterDrop();
        writeRouterReRoute();
        writePrimaryLinkActivity();
        writeSecondaryLinkActivity();
    }

```

```java

    /**
     * write the router drop fraction plot
     * @throws IOException if it can't write to the file specified
     */
    private void writeRouterDrop() throws IOException {
        Plot routerDropFraction = new Plot()
        .plotTitle("Router Drop Fraction")
        .xAxisTitle ("Mean arrival rate (pkt/sec)")
        .yAxisTitle ("Drop fraction")
        .yAxisStart (0.0)
        .yAxisEnd (1.0)
        .yAxisTickFormat (new DecimalFormat ("0.0"))
        .seriesDots(null)
        .seriesColor(Color.RED)
        .xySeries(aDrop)
        .seriesColor(Color.ORANGE)
        .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
                Color.ORANGE, 7))
        .xySeries(bDrop)
        .seriesDots(null)
        .seriesColor(Color.GREEN)
        .xySeries(cDrop)
        .seriesColor(Color.BLUE)
        .xySeries(dDrop)
        .labelColor(Color.RED)
        .label("<b>A</b>", 42.5, .85)
        .labelColor(Color.ORANGE)
        .label("<b>B</b>", 42.5, .75)
        .labelColor(Color.GREEN)
        .label("<b>C</b>", 42.5, .65)
        .labelColor(Color.BLUE)
        .label("<b>D</b>", 42.5, .55);
        Plot.write(routerDropFraction, routerDropFile);
    }

    /**
     * write the primary link activity plot
     * @throws IOException if it can't write to the file specified
     */
    private void writePrimaryLinkActivity() throws IOException {
        Plot linkActivity = new Plot()
        .plotTitle("Primary Link Activity")
        .xAxisTitle ("Mean arrival rate (pkt/sec)")
        .yAxisTitle ("Link Activity Fraction")
        .yAxisStart (0.0)
        .yAxisEnd (1.0)
        .yAxisTickFormat (new DecimalFormat ("0.0"))
        .seriesDots(null)
        .seriesColor(Color.RED)
        .xySeries(adActivity)
        .seriesColor(Color.ORANGE)
        .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
                Color.ORANGE, 7))
        .xySeries(bdActivity)
        .seriesDots(null)
        .seriesColor(Color.GREEN)
        .xySeries(cdActivity)
        .seriesColor(Color.BLUE)
        .xySeries(d2Activity)
        .labelColor(Color.RED)
        .label("<b>A</b>", 42.5, .65)
        .labelColor(Color.ORANGE)
        .label("<b>B</b>", 42.5, .55)
        .labelColor(Color.GREEN)
        .label("<b>C</b>", 42.5, .45)
        .labelColor(Color.BLUE)
        .label("<b>D</b>", 42.5, .35);
        Plot.write(linkActivity, primaryActivityFile);
    }

    /**
     * write the secondary link activity plot
     * @throws IOException if it can't write to the file specified
     */
    private void writeSecondaryLinkActivity() throws IOException {
        Plot linkActivity = new Plot()
        .plotTitle("Secondary Link Activity")
        .xAxisTitle ("Mean arrival rate (pkt/sec)")
        .yAxisTitle ("Link Activity Fraction")
        .yAxisStart (0.0)
        .yAxisEnd (1.0)
        .yAxisTickFormat (new DecimalFormat ("0.0"))
        .seriesDots(null)
        .seriesColor(Color.RED)
        .xySeries(abActivity)
        .xySeries(acActivity)
        .seriesColor(Color.ORANGE)
        .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
                Color.ORANGE, 7))
        .xySeries(bcActivity)
        .xySeries(baActivity)
        .seriesColor(Color.GREEN)
        .seriesDots(null)
        .xySeries(caActivity)
        .xySeries(cbActivity)
        .seriesColor(Color.BLUE)
        .xySeries(daActivity)
        .xySeries(dbActivity)
        .xySeries(dcActivity)
        .labelColor(Color.RED)
        .label("<b>A</b>", 42.5, .45)
        .labelColor(Color.ORANGE)
        .label("<b>B</b>", 42.5, .35)
        .labelColor(Color.GREEN)
        .label("<b>C</b>", 42.5, .25)
        .labelColor(Color.BLUE)
        .label("<b>D</b>", 42.5, .15);
        Plot.write(linkActivity, secondaryActivityFile);
    }

    /**
     * write the router re-route fraction plot
     *
     * @throws IOException if it can't write to the file specified
     */
    private void writeRouterReRoute() throws IOException {
        Plot reRouteFraction = new Plot()
        .plotTitle("Router Re-Route Fraction")
        .xAxisTitle ("Mean arrival rate (pkt/sec)")
        .yAxisTitle ("Re-Route fraction")
        .yAxisStart (0.0)
        .yAxisEnd (1.0)
        .yAxisTickFormat (new DecimalFormat ("0.0"))
        .seriesDots(null)
        .seriesColor(Color.RED)
        .xySeries(aReRoute)
        .seriesColor(Color.ORANGE)
        .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
                Color.ORANGE, 7))
        .xySeries(bReRoute)
        .seriesDots(null)
        .seriesColor(Color.GREEN)
        .xySeries(cReRoute)
        .seriesColor(Color.BLUE)
        .xySeries(dReRoute)
        .labelColor(Color.RED)
        .label("<b>A</b>", 42.5, .55)
        .labelColor(Color.ORANGE)
        .label("<b>B</b>", 42.5, .45)
        .labelColor(Color.GREEN)
        .label("<b>C</b>", 42.5, .35)
        .labelColor(Color.BLUE)
        .label("<b>D</b>", 42.5, .25);
        Plot.write(reRouteFraction, reRouteFile);
    }

    /**
     * Save the plot information into files.
     *
     * @param titlePrefix Prefix of the plot's title
     * @param yFormat decimal format of the traversal time y-axis labels
     * @param df drop fraction series
     * @param dfFile drop fraction file name
     * @param rt response time series
     * @param rtFile response time file
     * @throws IOException if it fails to write to any of the specified files
     */
    private void write(String titlePrefix, String yFormat, ListXYSeries df,
            String dfFile, ListXYSeries rt, String rtFile) throws IOException {
        Plot responseTime = new Plot()
        .plotTitle (titlePrefix+" Traversal Time")
        .xAxisTitle ("Mean arrival rate (pkt/sec)")
        .yAxisTitle ("Mean traversal time (sec)")
        .yAxisTickFormat (new DecimalFormat (yFormat))
        .seriesDots (null)
        .xySeries (rt);
        Plot dropFraction = new Plot()
        .plotTitle (titlePrefix+" Drop Fraction")
        .xAxisTitle ("Mean arrival rate (pkt/sec)")
        .yAxisTitle ("Drop fraction")
        .yAxisStart (0.0)
        .yAxisEnd (1.0)
        .yAxisTickFormat (new DecimalFormat ("0.0"))
        .seriesDots (null)
        .xySeries (df);
        Plot.write(responseTime, rtFile);
        Plot.write(dropFraction, dfFile);
    }

    /**
     * Open a GUI for each plot in order to visualize the results of a
     * previously run set of simulations.
     *
     * @param args each plot file generated that you wish to visualize
     */
    public static void main(String args[]) {
        if(args.length < 1) {
            System.err.println("Must specify at least 1 plot file.");
            usage();
        }

        for(int i = 0; i < args.length; i++) {
            try {
                Plot plot = Plot.read(args[i]);
                plot.getFrame().setVisible(true);
            } catch (ClassNotFoundException e) {
                System.err.println("Could not deserialize " + args[i]);
            } catch (IOException e) {
                System.err.println("Could not open " + args[i]);
            } catch (IllegalArgumentException e) {
                System.err.println("Error in file " + args[i]);
            }
        }
    }

    /**
     * Print the usage message for this program and gracefully exit.
     */
    private static void usage() {
        System.err.println("usage: java PlotHandler <plot-file-1> "+
                "(<plot-file-2> <plot-file-3>... etc.)");
        System.exit(1);
    }
}

```

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

import edu.rit.sim.Event;
import edu.rit.sim.Simulation;

/**
 * Class Routable is the abstract base class that defines objects that contain
 * routing logic with the ability to be linked together. Known implementations
 * include Router and Host.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-6-2015
 */
public abstract class Routable {

    private static int count = 0;

    /**
     * the simulation reference
     */
    protected final Simulation sim;
    private final int id;


    /**
     * Construct a routable object
     * @param sim the simulation this object should belong to
     */
    public Routable(Simulation sim) {
        this.sim = sim;
        id = ++ count;
    }

    /**
     * compare this instance with another object and determine whether this
     * instance is equal to the other object.
     *
     * @param o the other object to compare to
     * @return true if this object is equal to the other object
     */
    public boolean equals(Object o) {
        if(o == this) {
            return true;
        }
        if(o instanceof Routable) {
            return this.id == ((Routable)o).id;
        }
        return false;
    }

    /**
     * Called when this routable object finished receiving a packet on a certain
     * link
     * @param packet the packet this object received
     * @param link the link that the packet was received on
     */
    public abstract void receivePacket(final Packet packet, final Link link);

    /**
     * Send a given packet along a given link to another routable
     *
     * @param packet the packet to send
     * @param link the link to send the packet along
     */
    public void startSending(final Packet packet, final Link link) {
        final Routable other = link.other(this);
        final double transmitTime = packet.transmitTime(link);
        link.close();
        sim.doAfter(transmitTime, new Event() {
            public void perform() {
                other.receivePacket(packet, link);
            }
        });
    }
}
```

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

import edu.rit.sim.Simulation;
import edu.rit.util.AList;
import edu.rit.util.Random;

/**
 * Class models a router's behavior where packets are transmitted on a
 * preferred link if that link is available, otherwise a secondary link is
 * chosen at random until an available link is found. If no secondary links
 * are available, the packet is dropped.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-2-2015
 */
public class Router extends Routable {

    // private data members

    private final Random prng;
    private Link primary;
    private int dropCount;
    private int receiveCount;
    private int reRouteCount;
    private final AList<Link> secondary;

    /**
     * Construct a router object
     *
     * @param prng the pseudorandom number generator to use for choosing what
     * secondary routables to use
     * @param sim the simulation object this router should be associated with
     */
    public Router(Random prng, Simulation sim) {
        super(sim);
        this.prng = prng;
        this.dropCount = 0;
        this.receiveCount = 0;
        this.reRouteCount = 0;
        this.secondary = new AList<Link>();
    }

    /**
     * Set the primary link this router should prefer to send its received
     * packets on
     *
     * @param link the link to prioritize
     */
    public void setPrimary(Link link) {
        this.primary = link;
    }

    /**
     * add a secondary link to the list of secondary links
     *
     * @param link the link to add
     */
    public void addSecondary(Link link) {
        this.secondary.addLast(link);
    }

    /**
     * Called when this routable object finished receiving a packet on a certain
     * link
     * @param packet the packet this object received
     * @param link the link that the packet was received on
     */
    public void receivePacket(final Packet packet, final Link l) {
        l.open();
        Link link = null;
        ++receiveCount;
        boolean goodToGo = false;
        if(primary.ready()) {
            goodToGo = true;
            link = primary;
        } else if(secondary.size() > 0) {

            int[] indices = ShuffleHelper.shuffledArray(prng, secondary.size());
            for(int i = 0; i < indices.length && !goodToGo; i++) {
                link = secondary.get(indices[i]);
                if(link.ready()) {
                    goodToGo = true;
                    ++reRouteCount;
                }
            }
        }
        if(goodToGo) {
            startSending(packet, link);
        } else {
            // drop packet
            ++dropCount;
        }
    }

    /**
     * Get the fraction of packets that this router dropped
     *
     * @param totalPacketCount the total number of packets generated in the
     * simulation
     * @return a number between 0 and 1
     */
    public double dropFraction(int totalPacketCount) {
        return ((double)this.dropCount)/(double)totalPacketCount;
    }

    /**
     * Get the fraction of the packets that the router had to re-route along
     * a secondary route
     */
    public double reRouteFraction() {
        return receiveCount == 0 ? 0 :
            ((double)reRouteCount)/(double)receiveCount;
    }
}

```

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

import edu.rit.util.Random;

/**
 * Class provides helper methods for picking secondary links to transmit
 * packets on when primary links are currently in use.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-3-2015
 */
public class ShuffleHelper {

    /**
     * Shuffle an array in place
     *
     * @param prng pseudorandom number generator used with shuffling
     * @param array the array to shuffle
     */
    private static void shuffleArray(Random prng, int[] array) {
        for (int i = array.length - 1; i > 0; i--) {
            int index = prng.nextInt(i + 1);
            int a = array[index];
            array[index] = array[i];
            array[i] = a;
        }
    }

    /**
     * Create an array with <I>size</I> elements ranging from 0 to
     * <I>size - 1</I>.
     *
     * @param size the number of elements the array should contain
     * @return the array containing elements from 0 to <I>size - 1</I>
     */
    private static int[] indexArray(int size) {
        int[] retval = new int[size];
        for(int i = 0; i < size; i++) {
            retval[i] = i;
        }
        return retval;
    }

    /**
     * Create a shuffled array with <I>size</I> elements ranging from 0 to
     * <I>size - 1</I>.
     *
     * @param prng pseudorandom number generator used for shuffling
     * @param size number of elements to contain
     * @return the shuffled array
     */
    public static int[] shuffledArray(Random prng, int size) {
        int[] arr = indexArray(size);
        shuffleArray(prng, arr);
        return arr;
    }
}

```