**Source Code:**

**Animator.java**

package animatorPackage;  
  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.InputStream;  
import java.net.URL;  
import javax.swing.\*;  
  
// Represents an animator application for simulating transmission delay versus propagation delay  
public class Animator extends JFrame {  
 private JButton startButton = new JButton("Start");  
 private JButton resetButton = new JButton("Reset");  
  
 // Selector for length, rate, and packet size parameters  
 private ParameterSelector lengthSelector = new ParameterSelector(  
 new String[] { "10 km", "100 km", "500 km", "1000 km" },  
 new double[] { 10E3, 100E3, 500E3, 1E6 }, 1  
 );  
 private ParameterSelector rateSelector = new ParameterSelector(  
 new String[] { "512 kps", "1 Mbps", "10 Mbps", "100 Mbps" },  
 new double[] { 512E3, 1E6, 10E6, 100E6 }, 1  
 );  
 private ParameterSelector sizeSelector = new ParameterSelector(  
 new String[] { "100 Bytes", "500 Bytes", "1 kBytes" },  
 new double[] { 8E2, 4E3, 8E3 }, 1  
 );  
  
 private Thread simulationThread;  
 private SimulationTask simulationTask;  
 private boolean pktrun = false;  
   
 // Main method to start the animator application  
 public static void main(String[] args) {  
 Animator app = new Animator();  
 app.setVisible(true);  
 }  
  
 // Represents a parameter selector choice with associated values  
 private static class ParameterSelector extends Choice {  
 private double values[];  
  
 // Constructor to initialize the selector with items, values, and default selection  
 public ParameterSelector(String items[], double values[], int defaultValue) {  
 for (int i = 0; i < items.length; i++) {  
 super.addItem(items[i]);  
 Font font = new Font("Arial", Font.BOLD, 12);  
 super.setFont(font);  
 }  
 this.values = values;  
 select(defaultValue - 1);  
 }  
  
 // Returns the selected value from the selector  
 public double getSelectedValue() {  
 return values[super.getSelectedIndex()];  
 }  
 }  
  
 // Handles resource loading for images  
 final public static class ResourceLoader {  
 public static InputStream load(String path) {  
 InputStream input = ResourceLoader.class.getResourceAsStream(path);  
 if (input == null) {  
 input = ResourceLoader.class.getResourceAsStream("/" + path);  
 }  
 return input;  
 }  
 }  
  
 private CommunicationLink communicationLink;  
  
 // Initializes the animator application  
 public Animator() {  
 setTitle("Transmission delay versus propagation delay");  
 setSize(700, 200);  
 setLocationRelativeTo(null);  
 setBackground(Color.WHITE);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 JPanel contentPane = new JPanel();  
 contentPane.setLayout(new BorderLayout());  
  
 // Panel for parameter selection  
 JPanel primaryPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 10));  
 Font labelFont = new Font("Calibri", Font.BOLD, 15);  
 primaryPanel.add(new JLabel("Length", JLabel.RIGHT)).setFont(labelFont);  
 primaryPanel.add(lengthSelector);  
 primaryPanel.add(new JLabel("Rate", JLabel.RIGHT)).setFont(labelFont);  
 primaryPanel.add(rateSelector);  
 primaryPanel.add(new JLabel("Packet size", JLabel.RIGHT)).setFont(labelFont);  
 primaryPanel.add(sizeSelector);  
  
 // Communication link for visualization  
 communicationLink = new CommunicationLink(80, 22, 510, 18);  
 JPanel mainPanel = new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 communicationLink.drawCommunicationLink(g);  
  
 // Draws images for sender and receiver  
 drawImage(g, "/animatorPackage/icon.png", 15, 14, 60, 35, "Sender", 17, 65);  
 drawImage(g, "/animatorPackage/icon.png", 597, 14, 60, 35, "Receiver", 595, 65);  
  
 // Displays propagation speed information  
 g.drawString("Propagation speed : 2.8 x 10^8 m/sec", 160, 90);  
 }  
 };  
  
 primaryPanel.setBackground(Color.WHITE);  
 mainPanel.setBackground(Color.WHITE);  
 contentPane.add(mainPanel, BorderLayout.CENTER);  
  
 Font buttonFont = new Font("Arial", Font.BOLD, 12);  
 startButton.setFont(buttonFont);  
 resetButton.setFont(buttonFont);  
   
 // Disables focus painting for buttons  
 startButton.setFocusPainted(false);  
 resetButton.setFocusPainted(false);  
  
 // ActionListener for start button  
 startButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent event) {  
 launchSimulation();  
 }  
 });  
 primaryPanel.add(startButton);  
  
 // ActionListener for reset button  
 resetButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent event) {  
 stopSimulation();  
 communicationLink.updateTime(0);  
 mainPanel.repaint();  
 }  
 });  
 primaryPanel.add(resetButton);  
  
 contentPane.add(primaryPanel, BorderLayout.NORTH);  
  
 setContentPane(contentPane);  
 }  
  
 // Launches the simulation  
 private void launchSimulation() {  
 setComponentsEnabled(false);  
 communicationLink.configure(lengthSelector.getSelectedValue(), rateSelector.getSelectedValue());  
 communicationLink.sendDataPacket(sizeSelector.getSelectedValue(), 0);  
 simulationTask = new SimulationTask(1E-5, communicationLink.getTotalTime());  
 simulationThread = new Thread(simulationTask);  
 pktrun = true;  
 simulationThread.start();  
 }  
  
 // Stops the simulation  
 private void stopSimulation() {  
 if (simulationTask != null) {  
 simulationTask.endNow();  
 }  
 rootPaneCheckingEnabled = false;  
 setComponentsEnabled(true);  
 }  
  
  
 // Sets the enabled state of components  
 private void setComponentsEnabled(boolean value) {  
 startButton.setEnabled(value);  
 lengthSelector.setEnabled(value);  
 rateSelector.setEnabled(value);  
 sizeSelector.setEnabled(value);  
 }  
  
 // Draws an image with specified details  
 private void drawImage(Graphics g, String imagePath, int x, int y, int width, int height, String text, int textX, int textY) {  
 URL imgURL = ResourceLoader.class.getResource(imagePath);  
 Image img = Toolkit.getDefaultToolkit().getImage(imgURL);  
 g.drawImage(img, x, y, width, height, this);  
 g.setColor(Color.BLACK);  
 Font font = new Font("Calibri", Font.BOLD, 15);  
 g.setFont(font);  
 g.drawString(text, textX, textY);  
 }  
  
 // Represents a simulation task for animation  
 class SimulationTask implements Runnable {  
 private double simulationTimeCounter;  
 private double simulationLength;  
 private double simulationTick;  
  
 // Initializes the simulation task with tick and length  
 public SimulationTask(double tick, double length) {  
 simulationTick = tick;  
 simulationLength = length;  
 simulationTimeCounter = 0;  
 }  
  
 // Runs the simulation task  
 public void run() {  
 while (Animator.this.pktrun) {  
 simulationTimeCounter += simulationTick;  
 Animator.this.communicationLink.updateTime(simulationTimeCounter);  
 Animator.this.repaint();  
 if (simulationTimeCounter >= simulationLength) {  
 Animator.this.communicationLink.clearDataPackets();  
 Animator.this.simulationThread.suspend();  
 }  
 try {  
 Animator.this.simulationThread.sleep(50);  
 } catch (Exception e) {  
 // Handle exception   
 }  
 }  
 }  
  
 // Ends the simulation task  
 public void endNow() {  
 simulationLength = simulationTimeCounter;  
 }  
 }  
  
}

**CommunicationLink.java**

package animatorPackage;  
import java.awt.\*;  
  
// Represents a communication link with graphical and simulation properties  
class CommunicationLink {  
 private int positionX;  
 private int positionY;  
 private int linkWidth;  
 private int linkHeight;  
  
 final double WAVE\_PROPAGATION\_SPEED = 2.8E+8;  
 private double linkLength;  
 private double dataTransmissionRate;  
  
 private double currentSimulationTime;  
 private DataPacket currentDataPacket;  
  
 // Constructor to initialize the communication link with position and size  
 public CommunicationLink(int x, int y, int width, int height) {  
 positionX = x;  
 positionY = y;  
 linkWidth = width;  
 linkHeight = height;  
 }  
  
 // Configures the communication link with length and data transmission rate  
 public void configure(double length, double rate) {  
 linkLength = length;  
 dataTransmissionRate = rate;  
 }  
  
 // Updates the simulation time and removes data packets that have exceeded transmission time  
 void updateTime(double simulationTime) {  
 currentSimulationTime = simulationTime;  
 removeReceivedDataPackets(simulationTime);  
 }  
  
 // Sends a data packet with specified size and emission time  
 void sendDataPacket(double size, double packetEmissionTime) {  
 currentDataPacket = new DataPacket(size, packetEmissionTime);  
 }  
  
 // Removes data packets that have exceeded transmission time  
 private void removeReceivedDataPackets(double simulationTime) {  
 if (!(currentDataPacket == null)) {  
 if (simulationTime > currentDataPacket.packetEmissionTime + (currentDataPacket.dataSize / dataTransmissionRate) + linkLength \* WAVE\_PROPAGATION\_SPEED) {  
 clearDataPackets();  
 }  
 }  
 }  
  
 // Clears the current data packet  
 public void clearDataPackets() {  
 currentDataPacket = null;  
 }  
  
 // Calculates the total time for the communication link  
 public double getTotalTime() {  
 double packetEmissionTime = (currentDataPacket.dataSize / dataTransmissionRate);  
 double onLinkTime = (linkLength / WAVE\_PROPAGATION\_SPEED);  
 return (packetEmissionTime + onLinkTime);  
 }  
  
 // Draws the communication link and data packets  
 public void drawCommunicationLink(Graphics graphics) {  
 graphics.setColor(Color.white);  
 graphics.fillRect(positionX, positionY + 1, linkWidth, linkHeight - 2);  
 graphics.setColor(Color.black);  
 graphics.drawRect(positionX, positionY, linkWidth, linkHeight);  
 graphics.setColor(new Color(255, 182, 193)); // Pink color  
 Font font = new Font("Times New Roman", Font.BOLD, 17);  
 graphics.setFont(font);  
 graphics.drawString(convertTimeToString(currentSimulationTime), positionX + linkWidth / 2 - 30, positionY + linkHeight + 20);  
 drawDataPackets(graphics);  
 }  
  
 // Draws data packets on the communication link  
 private void drawDataPackets(Graphics graphics) {  
 if (!(currentDataPacket == null)) {  
 double timeRelativeToEmission = currentSimulationTime - currentDataPacket.packetEmissionTime;  
 double packetStartTime = timeRelativeToEmission - (currentDataPacket.dataSize / dataTransmissionRate);  
 double packetEndTime = timeRelativeToEmission;  
 packetStartTime = packetStartTime \* WAVE\_PROPAGATION\_SPEED \* linkWidth / linkLength;  
 packetEndTime = packetEndTime \* WAVE\_PROPAGATION\_SPEED \* linkWidth / linkLength;  
 if (packetStartTime < 0) {  
 packetStartTime = 0;  
 }  
 if (packetEndTime > linkWidth) {  
 packetEndTime = linkWidth;  
 }  
 graphics.setColor(Color.red);  
 graphics.fillRect(positionX + (int) (packetStartTime), positionY + 1, (int) (packetEndTime - packetStartTime), linkHeight - 1);  
 }  
 }  
  
 // Converts simulation time to string format  
 private static String convertTimeToString(double simulationTime) {  
 String result = Double.toString(simulationTime \* 1000);  
 int dotIndex = result.indexOf('.');  
 String decimalPart = result.substring(dotIndex + 1) + "000";  
 decimalPart = decimalPart.substring(0, 3);  
 String integerPart = result.substring(0, dotIndex);  
 return integerPart + "." + decimalPart + " ms";  
 }  
}  
  
// Represents a data packet with size and emission time  
class DataPacket {  
 double dataSize;  
 double packetEmissionTime;  
  
 // Constructor to initialize data packet with size and emission time  
 DataPacket(double size, double emissionTime) {  
 this.dataSize = size;  
 this.packetEmissionTime = emissionTime;  
 }  
}

***Code Description:***

The Animator Class extends to JFrame. It has Variables defined for Buttons and drop-downs for Length, Packet Size, and Transmission rate.

The Animator constructor has the JPanel which acts as a Top-level container. It contains the Main User interface design. We have declared two JButtons Start and Stop which helps to start the animation with the selected values of Length, Packet Size, and Rate. When we click the Reset button it will stop the animation and we will see the Total Time taken by Packet to reach the destination from the source.

There are two action listeners defined for the two buttons to start and stop the animation. On clicking the start button the Launch simulation method is called and clicking the reset button the Stop simulation method is called which stops the animations.

The CommunicationLink class contains the methods to create a Communication channel with the values that are passed from the animator class. This class contains methods that calculate the total simulation Time. It contains methods that are used to draw the Data packet and to display the animation from sender to receiver for a specific time.