public class **CommunicationEnvironment.Cell**

This class handles general cell structure with zone informations. It also handles deployment of nodes in a specified zone or entire cell.

|  |  |
| --- | --- |
| Constructors | **public Cell(**  **CRBase baseStation,**  **double radius,**  **int number\_of\_sectors,**  **int alpha,**  **ArrayList set\_of\_d)**  Constructor of the Cell  **Parameters**  baseStation - CRBase  radius - Radius of the Cell  number\_of\_sectors - Number of sectors in the cell.  alpha - Corresponding angle for a zone at the baseStation  set\_of\_d - List of distances |
| Methods | **public static java.awt.geom.Point2D.Double deployNodeinCell()**  Finds a random position for a node in the Cell.  **Returns**  Position of the node.  **public static java.awt.geom.Point2D.Double deployNodeinZone(**  **int sector\_number,**  **int angle\_number,**  **int distance\_number)**  Finds a random position for a node in a specified zone.  Note: All the parameter values starts from zero.  **Parameters**  sector\_number - Sector number of the node.  angle\_number - Angle number in the sector of the node.  distance\_number - Distance of the zone to the center.  **Returns**  Position of the node.  **public static java.awt.geom.Point2D.Double deployNodeInRouteCircle(**  **Node node,**  **double routeRadius)**  Deploys a node within a range it could have been relocated. This method is designed for primary node relocation only. It is assured that the node will never leave the cell.  **Parameters**  node Node - to relocate  routeRadius Range - of node's relocation  **Returns**  New location of the node  **public static void setBaseStation(**  **CRBase baseStation)**  Sets a CRBase as a base station to the Cell.  **Parameters**  baseStation - Base station  **public static CRBase getBaseStation()**  Gets the current baseStation of the Cell.  **Returns**  Base Station  **public void setPosition(**  **Point2D.Double position)**  Sets a new position for the baseStation.  **Parameters**  -  **public java.awt.geom.Point2D.Double getPosition()**  Gets the current position of the baseStation.  **Returns**  Position of the baseStation  **public static void setAlpha(**  **int alpha)**  Sets a new alpha value for the Cell.  **Parameters**  alpha - Alpha  **public static int getAlpha()**  Gets the current alpha value of the Cell.  **Returns**  Alpha  **public static void setNumber\_of\_sectors(**  **int number\_of\_sectors)**  Sets a new number of sectors value for the Cell.  **Parameters**  number\_of\_sectors - Number of sectors in the Cell  **public static int getNumber\_of\_sectors()**  Gets the current number of sectors value of the Cell.  **Returns**  Number of sectors  **public static void setRadius(**  **double radius)**  Sets a new radius value for the Cell.  **Parameters**  radius - Radius of the Cell.  **public static double getRadius()**  Gets the current radius value of the Cell.  **Returns**  Radius of the Cell  **public static void setSet\_of\_d(**  **ArrayList set\_of\_d)**  Sets a new distance list for the Cell.  **Parameters**  set\_of\_d - Distance list  **public static java.util.ArrayList getSet\_of\_d()**  Gets the current distance list of the Cell.  **Returns**  Distance list |
| Fields | **public static set\_of\_d**  List of all distances between the baseStation and the zones which are in the same sector and have the same angle interval with the baseStation. Distances must be in the ascending order. |

public class **Nodes.CRBase** extends Node

This class handles basic operation of a CR base station such as sensing schedule advertisement, communication schedule advertisement, handoff of CR users, etc. It also keeps information about which CR node belongs which zone

|  |  |
| --- | --- |
| Constructors | **public CRBase(**  **Point2D.Double pos,**  **int id,**  **int number\_of\_freq\_per\_crnode)**  Creates a CRBase at the given position.  **Parameters**  pos - Position of the Base station  id - Id of the CRBase  number\_of\_freq\_per\_crnode - The number of the frequencies that a CRNode is going to listen in a frame. |
| Methods | **public java.util.ArrayList deploy\_freq(**  **boolean startFromFirst)**  Takes number\_of\_freq\_per\_crnode frequencies from frequency list by paying attention to the order of the frequencies.  **Parameters**  startFromFirst - If true the deployed frequency values will start from the first frequency  **Returns**  number\_of\_freq\_per\_crnode frequencies  **public void assignFrequencies()**  Assigns frequencies to the crnodes to be listened in the sensing slots and also updates frequency list.  **public void communicationScheduleAdvertiser()**  First, finds the threshold value for the collision purposes. Second, finds available(free) frequencies by using threshold value. Third, deploys these free frequencies to the crnodes to communicate at the next frame.  **public void setLast\_averageSnr(**  **ArrayList current\_averageSnr)**  Updates average\_snr values. Assigns the previous current\_averagesnr value to the last\_averagesnr.  **Parameters**  current\_averageSnr - Most up-to-date snr value.  **public java.util.ArrayList getFrequency\_list()**  Returns the frequency\_list  **Returns**  frequency\_list  **public void registerZone(**  **int sector,**  **int alpha,**  **int d,**  **int crnodes)**  Takes parameters of a zone and registers this zone into registeredZones  **Parameters**  sector - Sector of the zone  alpha - Alpha number of the zone  d - Distance section of the zone  crnodes - Total number of crnodes in that zone  **public java.awt.geom.Point2D.Double deployNodeinZone(**  **int id)**  Takes id of crnode then finds that crnode's zone, after this, calls deployNodeinZone function with the corresponding zone parameters.  **Parameters**  id - Id of CRNode  **Returns**  Point of the CRNode  **public int findZone(**  **int id)**  Finds zone ID of a given CR node  **Parameters**  id ID - of CR node  **Returns**  Zone ID of the given CR node  **public double farthestZoneDistance()**  Finds farthest point of the all registered zones to base station  **Returns**  Distance of the farthest point |
| Fields | **public registeredZones**  Keeps sector number, alpha number, d number, and number of CR nodes of a zone  **public static final SECTOR**  Index of sector number in zone array  **public static final ALPHA**  Index of alpha number in zone array  **public static final D**  Index of d number in zone array  **public static final CRNODES**  Index of number of CR nodes in zone array |

public class **DESSimulation.CRDESScheduler** extends SimEnt

This class handles the frame structure of the CR nodes.

|  |  |
| --- | --- |
| Constructors | **public CRDESScheduler(**  **double simulationDuration,**  **double unitTime,**  **int numberOfSlots,**  **double slotDur,**  **double senseScheduleAdvertisement,**  **double commScheduleAdvertisement,**  **double commDur,**  **double senseResultAdvertisement)**  Creates a DES scheduler that performs frame action for CR sensor nodes  **Parameters**  simulationDuration Duration - of the simulation in unit time  unitTime Unit - of time in milliseconds  numberOfSlots Number - of sensing slots in the frame  slotDur Duration - of the sensing slots in terms of unit time  senseScheduleAdvertisement Duration - of the sensing schedule advertisement in terms of unit time  commScheduleAdvertisement Duration - of the communication schedule advertisement in terms of unit time  commDur Duration - of the communication in terms of unit time  senseResultAdvertisement Duration - of the sensing result advertisement in terms of unit time |
| Methods | **public void start()**  Starts the Simulation  **public void recv(**  **SimEnt src,**  **Event ev)**  Main frame operation  **Parameters**  src Source - of the event  ev Occured - Event  **protected void destructor()**  Sets and displays statistics about simulation and reset the GUI on the program.  **public void terminate()**  Terminates the simulation  **public boolean isFinished()**  Returns whether the simulation is finished or not.  **Returns**  True if simulation is finished, false otherwise.  **public void deliveryAck(**  **EventHandle h)**  Acknowledge event delivery  **Parameters**  h Handle - of the event  **public double getRemainingSimulationDuration()**  Returns remaining simulation duration  **Returns**  Remaining simulation duration  **public double getSimulationDuration()**  Returns total simulation duration  **Returns**  Total simulation duration  **public void sendEndCommEvent(**  **int crnode\_id)**  Sends a communication end event for a given CR node  **Parameters**  crnode\_id ID - of the CR node  **public void sendStartCommEvent(**  **int crnode\_id)**  Sends a communication start event for a given CR node  **Parameters**  crnode\_id ID - of the CR node |

public class **Nodes.CRNode** extends Node

This class basic operations of a CR node. It also concerns with logging operations communication events of all CR nodes.

**See Also**

Node

|  |  |
| --- | --- |
| Constructors | **public CRNode(**  **int id,**  **Point2D.Double pos,**  **double vel)**  Creates a CRNode with the given frequencies, position and velocity values.  **Parameters**  id - ID of this CR node  pos - Position of the CRNode  vel - Velocity of the CRNode |
| Methods | **public void sense(**  **int freq)**  Updates the snr value of the frequency.  **Parameters**  freq - Number of the frequency in snrValues list.  **public java.util.HashMap getSnrValues()**  **Returns**  Snr values of each frequencies which are assigned to this node.  **public static void initializeAverageSnr(**  **int total\_number\_of\_frequencies,**  **int numberOfZones)**  It creates the averageSnr arraylist and initially add zeros to the elements.  **Parameters**  total\_number\_of\_frequencies Total - number of frequencies  numberOfZones Number - of zones currently simulating  **public void logSnrValues()**  Writes the id of the CRNode, position of the CRNode and snrValues of the CRNode to the log file, respectively.  **public static void logAverageSnr(**  **double time)**  Calculates average snr values then writes these values to the log file and then resets the average snr values.  **Parameters**  time - Current time  **public static void createLogFile(**  **String file\_name)**  Creates the log file.  **Parameters**  file\_name - Name of the log file  **public static void writeLogFile(**  **String log\_string)**  Writes the input string to the log file.  **Parameters**  log\_string - String  **public static void closeLogFile()**  Closes the log file.  **public void setFrequencyList(**  **ArrayList frequencies)**  Sets frequency list to listen in the sensing slots.  **Parameters**  frequencies - Frequency list  **public void setCommunication\_frequency(**  **int communication\_frequency)**  Sets a frequency to communicate  **Parameters**  communication\_frequency - Communication frequency  **public int getCommunication\_frequency()**  Returns the current communication frequency of this CR node  **Returns**  Current communication frequency  **public void releaseCommunication\_frequency()**  Releases its communication frequency  **public static void communicate(**  **double time,**  **boolean lastReport)**  Checks whether a collision occured or not if there is an assigned frequency to the crnode for that frame and resets the collision value if it is the last report.  **Parameters**  time - Time of the simulation  lastReport - True if it is the last report otherwise false  **public static void setTotalNumberOfFrames(**  **int totalNumberOfFrames)**  Sets the total number of frames for a simulation.  **Parameters**  totalNumberOfFrames - Number of Frames  **public static int getTotalNumberOfFrames()**  Returns total number of frames.  **Returns**  Total number of frames  **public static java.lang.String[][] logStats()**  Calculates and writes CrNode statistics to the log file.  **Returns**  CrNode statistic values  **public double nextOnDurationDES(**  **double frameDuration)**  Finds the next on duration according to the traffic model for DES  **Parameters**  frameDuration Duration - of one frame  **Returns**  On duration  **public double nextOffDurationDES(**  **double frameDuration)**  Finds the next off duration according to the traffic model for DES  **Parameters**  frameDuration Duration - of one frame  **Returns**  Off duration  **public int nextOnDuration(**  **double frameDuration)**  Finds the next on duration in terms of number of frames according to the traffic model for Multithreaded Simulation  **Parameters**  frameDuration Duration - of one frame  **Returns**  On duration  **public int nextOffDuration(**  **double frameDuration)**  Finds the next off duration in terms of number of frames according to the traffic model for Multithreaded Simulation  **Parameters**  frameDuration Duration - of one frame  **Returns**  Off duration  **public boolean getCommOrNot()**  Returns whether this CR node is currently communicating or not  **Returns**  <ul> <li> <b><i>True</i></b> if node is currently communicating</li> <li> <b><i>False</i></b> otherwise</li> </ul>  **public void setReadytoComm(**  **boolean readytoComm)**  Sets whether this node can start communication in this frame or not  **Parameters**  readytoComm Indicates - this node can wants to start communicating in this frame if possible  **public boolean getReadytoComm()**  Returns whether this CR node wants to start communicating in this frame or not  **Returns**   * ***True*** if node wants to start communicating in this frame * ***False*** otherwise   **public boolean getIsCollided()**  Returns whether this CR node is collided in previous frame or not  **Returns**   * ***True*** if node collided in previous frame * ***False*** otherwise   **public void setIsCollided(**  **boolean iscollided)**  Sets whether this node is collided in this frame or not  **Parameters**  iscollided - Indicates this node collided in this frame or not  **public int getNumberOfForcedHandoff()**  Returns the number of force handoffs this node has made  **Returns**  Number of forced handoffs  **public void setNumberOfForcedHandoff(**  **int numberOfForcedHandoff)**  Sets the number of forced handoffs this node has made  **Parameters**  numberOfForcedHandoff Number - of forced handoffs  **public int getNumberOfDrops()**  Returns how many times this node has dropped  **Returns**  Number of drops  **public void setNumberOfDrops(**  **int numberOfDrops)**  Sets how many times this node dropped  **Parameters**  numberOfDrops Number - of drops  **public int getNumberOfBlocks()**  Returns how many times this node has blocked  **Returns**  Number of blocks  **public void setNumberOfBlocks(**  **int numberOfBlocks)**  Sets how many times this node blocked  **Parameters**  numberOfBlocks - Number of blocks |
| Fields | **public numberOfBlocks**  Count of how many times this CR node is blocked  **public final startCommEvent**  An event to start communication for this CR node  **public startEventHandle**  Handle of the start communication event of this CR node  **public final endCommEvent**  An event to end communication for this CR node  **public endEventHandle**  Handle of the end communication event of this CR node |

public static class **Nodes.CRNode.StartCommunicationEvent**

Event class to handle communication starts of CR nodes

|  |  |
| --- | --- |
| Constructors | **public CRNode.StartCommunicationEvent(**  **int crnode\_id)**  Constructor of this event  **Parameters**  crnode\_id ID - of the associated CR node |
| Methods | **public void entering(**  **SimEnt locale)**  Method to handle entering of this event to a simulation entity  **Parameters**  locale Simulation - entity that this event entering |
| Fields | **public id**  ID of the associated CR node |

public static class **Nodes.CRNode.EndCommunicationEvent**

Event class to handle communication end of CR nodes

|  |  |
| --- | --- |
| Constructors | **public CRNode.EndCommunicationEvent(**  **int crnode\_id)**  Constructor of this event  **Parameters**  crnode\_id ID - of the associated CR node |
| Methods | **public void entering(**  **SimEnt locale)**  Method to handle entering of this event to a simulation entity  **Parameters**  locale Simulation - entity that this event entering |
| Fields | **public id**  ID of the associated CR node |

public class **MultiThreadedSimulation.CRSensorThread** implements java.lang.Runnable

This class handles the frame structure of the CR nodes.

|  |  |
| --- | --- |
| Constructors | **public CRSensorThread(**  **double simulationDuration,**  **double unitTime,**  **int numberOfSlots,**  **double slotDur,**  **double senseScheduleAdvertisement,**  **double commScheduleAdvertisement,**  **double commDur,**  **double senseResultAdvertisement)**  Creates a thread that performs simulation action for CR sensor nodes  **Parameters**  simulationDuration Duration - of the simulation in unit time  unitTime Unit - of time in milliseconds  numberOfSlots Number - of sensing slots in the frame  slotDur Duration - of the sensing slots in terms of unit time  senseScheduleAdvertisement Duration - of the sensing schedule advertisement in terms of unit time  commScheduleAdvertisement Duration - of the communication schedule advertisement in terms of unit time  commDur Duration - of the communication in terms of unit time  senseResultAdvertisement Duration - of the sensing result advertisement in terms of unit time |
| Methods | **public void run()**  Main thread operation  **public boolean isFinished()**  Returns whether the thread is finished or not  **Returns**  finished  **public void terminate()**  Terminates the thread  **public double getRemainingSimulationDuration()**  Returns remaining simulation duration  **Returns**  Remaining simulation duration  **public double getSimulationDuration()**  Returns total simulation duration  **Returns**  total simulation duration  **public double getUnitTime()**  Return the ms per unit time  **Returns**  Unit time  **public double getFrameDuration()**  Returns the frame duration in terms of msec  **Returns**  Frame Duration  **public void setCommunationDuration(**  **int crnode\_id)**  Sets the ending frame of communication of the given CR node  **Parameters**  crnode\_id ID - of the CR node  **public void setInactiveDuration(**  **int crnode\_id,**  **boolean dropped)**  Sets the starting frame of communication of the given blocked or dropped CR node  **Parameters**  crnode\_id ID - of the CR node  dropped Indicates - whether the node dropped or not |

public class **DESSimulation.DESPrimaryTrafficGenerator**

This class holds primary traffic generation model and registered primary nodes. It also logs and prepares the statistics of the simulation related to primary nodes.

|  |  |
| --- | --- |
| Constructors | **public DESPrimaryTrafficGenerator()**  Creates a primary traffic generator with no node registered to it and with the given model. |
| Methods | **public void registerNode(**  **PrimaryTrafficGeneratorNode n)**  Registers a node and creates a Simulation entity for it to generate traffic  **Parameters**  n node - to be registered  **public void start()**  Starts the simulation  **public java.lang.String[][] logStats()**  Logs and creates statistics of the simulation related to primary nodes  **Returns**  Primary node statistics of the simulation |

public class **Animation.DrawArea** extends javax.swing.JPanel

This class handles the animation window.

|  |  |
| --- | --- |
| Constructors | **public DrawArea(**  **int cellRadius,**  **int numberOfSectors,**  **int numberOfAlpha,**  **int numberOfDSections,**  **int numberOfCrNodes,**  **int numberOfPriNodes)**  Creates an animation window.  **Parameters**  cellRadius Radius - of the cell  numberOfSectors Number - of sectors in the cell  numberOfAlpha Number - of alpha slices in a sector  numberOfDSections - Number of distance sections in a alpha slice  numberOfCrNodes Number - of CR nodes in the zone  numberOfPriNodes Number - of Primary nodes in the cell |
| Methods | **public void paintComponent(**  **Graphics g)**  **public void paint(**  **Graphics g)**  **public void paintPrimary(**  **Integer id,**  **PointColor p)**  This method paints a primary node with the given id, position, and color.  **Parameters**  id ID - of the primary node  p Position - of the CR node  **public void paintCR(**  **Integer id,**  **PointColor p)**  This method paints a CR node with the given id, position.  **Parameters**  id ID - of the CR node  p Position - of the CR node |

public class **Animation.DrawCell** implements java.lang.Runnable

This class initiates the animation window. It also provides an interface for drawing new nodes on the animation window or changing the properties of the currently existing ones.

|  |  |
| --- | --- |
| Constructors | **public DrawCell(**  **int radius,**  **int numberOfSectors,**  **int numberOfAlpha,**  **int numberOfDSections,**  **int numberOfCrNodes,**  **int numberOfPriNodes)**  Constructs a cell structure to paint  **Parameters**  radius Radius - of the cell  numberOfSectors Number - of sectors in the cell  numberOfAlpha Number - of alpha sections in a sector  numberOfDSections - Number of distance sections in a alpha slice  numberOfCrNodes Number - of CR nodes in the zone  numberOfPriNodes Number - of primary nodes in the cell |
| Methods | **public void run()**  **public void draw()**  Draws the initial cell  **public static void paintPrimaryNode(**  **Node n,**  **Color c)**  Adds a Primary node to the cell with given color  **Parameters**  n - Node to be added  c - Nodes color: RED for transmitting or BLACK for sleeping  **public static void paintCrNode(**  **Node n,**  **Color c)**  Adds a CR node to the cell  **Parameters**  n - Node to be added  c - Color of the node  **public boolean isFinished()**  Returns whether the thread is finished or not  **Returns**  finished  **public void terminate()**  Terminates the thread |

public abstract class **Nodes.Node**

An abstract class that holds general information about both CR and Primary nodes.

|  |  |
| --- | --- |
| Constructors | **public Node()** |
| Methods | **public java.awt.geom.Point2D.Double getPosition()**  Gets the current position of the Node  **Returns**  position of the node  **public double getVelocity()**  Gets the current velocity of the Node  **Returns**  Velocity of the node  **public void setPosition(**  **Point2D.Double position)**  Sets a new position for the node  **Parameters**  position - Position of the node  **public void setVelocity(**  **double velocity)**  Sets a new velocity for the node  **Parameters**  velocity - Velocity of the node  **public void setId(**  **int id)**  Sets an id for the node.  **Parameters**  -  **public int getId()**  Returns the current id.  **Returns**  Id of the node |
| Fields | **protected position**  Position of the node  **protected velocity**  Velocity of the node  **protected id**  Id o the Node |

public class **SimulationRunner.ParetoDistribution**

This class implements Pareto distribution by using Uniform distribution class of CERN package.

|  |  |
| --- | --- |
| Constructors | **public ParetoDistribution(**  **double shapeParameter,**  **double minValue,**  **RandomEngine randomEngine)**  Creates a Pareto distribution  **Parameters**  shapeParameter Shape - parameter of the distribution  minValue Minimum - value of the distribution  randomEngine Random - number generator  **public ParetoDistribution(**  **double meanValue,**  **RandomEngine randomEngine)**  Creates a Pareto distribution with given expected value.  Note: Minimum value of the distribution is assumed to be 1, and so mean value cannot be less than 2.  **Parameters**  meanValue - Expected value of the distribution  randomEngine Random - number generator |
| Methods | **public double cdf(**  **double x)**  Returns the cumulative distribution function  **Parameters**  x Point - to evaluate the value of the function  **Returns**  Cumulative distribution function  **public double nextDouble()**  Returns a random number from the distribution  **Returns**  A random number form the distribution  **public double pdf(**  **double x)**  Returns the probability density function  **Parameters**  x Point - to evaluate the value of the function  **Returns**  Probability density function  **public void setMinValue(**  **double minValue)**  Sets the minimum value of the distribution.  **Parameters**  minValue New - minimum value  **public void setShapeParameter(**  **double shapeParameter)**  Sets the shape parameter value of the distribution.  **Parameters**  shapeParameter New - shape parameter value |

public class **Animation.Plot**

This class plots simple graphs. It can plot all graphs on to same window or different windows. For plotting it uses jPlot.

|  |  |
| --- | --- |
| Constructors | **public Plot(**  **int numberOfXs,**  **ArrayList yPerX)**  Creates a new plotter object with no x value and y value.  **Parameters**  numberOfXs Number - of different x values  yPerX How - many y belongs to each x |
| Methods | **public boolean addPoint(**  **int xPos,**  **double xVal,**  **ArrayList yVals)**  Adds an x value and its corresponding y values. This methods insert values in a way that will keep x values array always sorted  **Parameters**  xPos Which - x value collection the values belong  xVal x - value to be added  yVals y - values to be added  **Returns**  true if values added false otherwise.  **public void plot(**  **String title,**  **int xPos,**  **int yPos)**  Creates the given x versus the given y value file to be used to plot its graph. This method returns immediately if the y values is not valid.  **Parameters**  title title - of the graph  xPos - x values to be plotted  yPos y - values to be plotted  **public void plot(**  **String title,**  **ArrayList xs,**  **ArrayList ys,**  **ArrayList names)**  Plots the specified graphs onto the same window. It can plot 36 different graphs onto the same window. It plots each of the graphs with different colors and data point shapes. That is, no two different graphs will have the same color and same data point shapes.  **Parameters**  title Title - of the graphs  xs x - values to plotted  ys y - values to plotted  names Names - of the graphs  **Throws**  -  **public void plotAll()**  Plots all available x versus y values onto different graphs on different windows  **public void plotAllX(**  **int xPos)**  Plots all available y values belongs to given x value onto different graphs on different windows  **Parameters**  xPos x - value to be plotted  **public void plotAll(**  **ArrayList names)**  This method plots corresponding y values of different x values on the same window. It plots all possible graphs.  **Parameters**  names Names - of the graphs that will be plotted on the same window |

public class **Animation.PointColor**

This class holds the position of a node on the animation screen. It also keeps its color and radius.

|  |  |
| --- | --- |
| Constructors | **public PointColor(**  **int x,**  **int y,**  **int r,**  **Color c)**  Constructs a PointColor with specied coordinate and color  **Parameters**  x - x coordinate of the point  y - y coordinate of the point  r - Radius of the point  c - Color of the point  **public PointColor(**  **Point2D.Double p,**  **int r,**  **Color c,**  **int unit)**  Constructs a PointColor from Point2D by scaling its coordinate and with a specified color  **Parameters**  p - Point2D to be converted  r - Radius of the point  c - Color of the point  unit - Scale unit |
| Methods | **public Animation.PointColor convertCoordinate(**  **int cellRadius)**  Convert this PointColor object from regular x-y coordinate system to JFrame x-y coordinate system  **Parameters**  cellRadius - Radius of the cell  **Returns**  Returns this object for cascading |

public class **Nodes.PrimaryTrafficGeneratorNode** extends Node

This class handles basic operations of Primary nodes.

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGeneratorNode(**  **Point2D.Double pos,**  **double vel,**  **int id)**  Constructor of the PrimaryTrafficGeneratorNode  **Parameters**  pos - Position of the node  vel - Velocity of the node  id - ID of this node |
| Methods | **public void setRandomPosition(**  **double offDuration)**  Sets a new position for the primary traffic generator node.  **Parameters**  offDuration - Previous off duration  **public int generateTraffic(**  **double offDuration)**  Finds a free frequency and occupies it. This method is synchronized. That is only one thread at a time can run it  **Parameters**  offDuration - Previous off duration  **Returns**  ID of the occupied frequency  **public int getCommunicationFreq()**  Returns the current communication frequency of this Primary node  **Returns**  Communication frequency  **public int getNumberOfCallAttempts()**  Returns how many times this Primary node attempted to communicate  **Returns**  Number of call attempts  **public int getNumberOfDrops()**  Returns how many times this Primary node is dropped  **Returns**  Number of drops  **public double getComunicationDuration()**  Returns the total communication duration of this primary node  **Returns**  Communication duration  **public void incrementTotalCommunicationDuration(**  **double commDur)**  Increments communication duration of this Primary node  **Parameters**  commDur Last - communication duration  **public double getRoutingRadius()**  Returns routing radius of a primary node  **Returns**  Routing radius  **public static java.lang.String[][] logStats(**  **HashMap registeredNodes)**  Logs output statistics of Primary nodes  **Parameters**  registeredNodes List - of registered Primary nodes  **Returns**  Primary node statistics values |

public class **MultiThreadedSimulation.PrimaryTrafficGenerator**

This class holds necessary semaphores and traffic model information for primary traffic generation. It also logs and prepares the statistics of the simulation related to primary nodes.

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGenerator()**  Creates a primary traffic generator with no node registered to it |
| Methods | **public void registerNode(**  **PrimaryTrafficGeneratorNode n)**  Registers a node and creates a thread for it to generate traffic  **Parameters**  n node - to be registered  **public void terminateAllThreads()**  Terminates all associated threads  **public java.lang.String[][] logStats()**  Logs and creates statistics of the simulation related to primary nodes  **Returns**  Primary node statistics of the simulation |

public class **DESSimulation.PrimaryTrafficGeneratorSimEnt** extends SimEnt

This class handles events related to one primary node's traffic generation.

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGeneratorSimEnt(**  **PrimaryTrafficGeneratorNode node,**  **double meanOnDuration,**  **double meanOffDuration)**  Creates a simulation entity for the given node with given probabilistic mean values. It checks the traffic model from DESPrimaryTrafficGenerator class.  **Parameters**  node Node - associated to this simulation entity  meanOnDuration <ul> <li><i>If - Poisson traffic model:</i> Expected value for duration of a call in terms of time units <li><i>If ON-OFF traffic model:</i> Expected value for duration of a ON period in terms of time units </ul>  meanOffDuration <ul> <li><i>If - Poisson traffic model:</i> Mean number of calls per unit time <li><i>If ON-OFF traffic model:</i> Mean OFF period duration of a node in terms of time units </ul> |
| Methods | **public void recv(**  **SimEnt src,**  **Event ev)**  Main event handler  **Parameters**  src Source - of the event  ev Occured - event  **public void deliveryAck(**  **EventHandle h)**  Acknowledge event delivery  **Parameters**  h Handle - of the event |

public class **MultiThreadedSimulation.PrimaryTrafficGeneratorThread** implements java.lang.Runnable

Thread responsible for generating an invidual primary traffic in the wireless channel.

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGeneratorThread(**  **PrimaryTrafficGeneratorNode n,**  **double meanOnDuration,**  **double meanOffDuration)**  Creates a primary traffic generator thread associated with node n  **Parameters**  n Primary - node associated with this thread  meanOnDuration <ul> <li><i>If - Poisson traffic model:</i> Expected value for duration of a call in terms of time units <li><i>If ON-OFF traffic model:</i> Expected value for duration of a ON period in terms of time units </ul>  meanOffDuration <ul> <li><i>If - Poisson traffic model:</i> Mean number of calls per unit time <li><i>If ON-OFF traffic model:</i> Mean OFF period duration of a node in terms of time units </ul> |
| Methods | **public void run()**  Main thread operation  **public boolean isFinished()**  Returns whether the thread is finished or not  **Returns**  finished  **public void terminate()**  Terminates the thread |

public class **SimulationRunner.SimulationRunner** extends javax.swing.JFrame

This is the main class of the software. It instantiates necessary classes and create GUI for taking input from the user.

|  |  |
| --- | --- |
| Constructors | **public SimulationRunner()**  Constructor of the class. This method initializes the GUI. |
| Methods | **public static void main(**  **String[] args)**  **Parameters**  args - the command line arguments  **public void startSimulation()**  Initializes the main simulation threads or schedulers  **public static void clear()**  Clears the data of the simulation |
| Fields | **public static wc**  Main wireless channel which all types of nodes are accessing  **public static cell**  Cognitive radio cell structure  **public static priTrafGen**  Primary traffic generator thread for wireless channel frequencies  **public static crSensor**  Thread responsible for frame structure of CR nodes  **public static crDesScheduler**  DES Framework event scheduler for constructing frame structure of CR nodes  **public static priTrafGenDes**  DES Framework Primary traffic generator for wireless channel frequencies  **public static crBase**  Base station of CR cell  **public static crNodes**  CR nodes which sense the wireless channel  **public static priTrafGenNodes**  Primary traffic generator nodes which cause traffic in wireless channel  **public static randEngine**  Random number generator for all random number generation operations in the simulation  **public static plot**  Plots the time versus average SNR and SINR values graphs  **public static runner**  Currently running SimulationRunner instance  **public static terminateSimulation**  Button to terminate an ongoing simulation  **public static progressBar**  Progress bar to show progress of the simulation  **public static animationOnButton**  **public static animationOffButton**  **public static plotOnButton**  **public static plotOffButton** |

public class **Animation.SimulationStatsTable** extends javax.swing.JFrame

This class displays the output statistics of the simulation.

|  |  |
| --- | --- |
| Constructors | **public SimulationStatsTable(**  **String[][] crStats,**  **String[][] priStats,**  **JFrame parent)**  Creates a statistic table for the simulation results. It displays two tables with CR node statistics and Primary node statistics on them.  **Parameters**  crStats CR - node statistics  priStats Primary - node statistics  parent Parent - of the this frame |

public class **CommunicationEnvironment.WirelessChannel**

This class handles main channel operations, such as SNR and SINR calculations. It also provides frequencies to primary and CR nodes.

|  |  |
| --- | --- |
| Constructors | **public WirelessChannel(**  **int channelModel,**  **int numberOfFrequencies,**  **double maxSNR,**  **double sinrThreshold,**  **double meanOffDuration,**  **double meanOnDuration,**  **int trafficModel,**  **double unitTime)**  Creates a wireless channel with the given model. It creates numberOfFrequencies amount frequency. Initially there is no node in the channel.  **Parameters**  channelModel 0 - for Simple ch., 1 for Lognormal ch.  numberOfFrequencies Number - of frequencies in the channel  maxSNR max - SNR value of the channel  sinrThreshold SINR - threshold for CR nodes to be able to communicate without collision  meanOffDuration - <ul> <li><i>If Poisson traffic model:</i> Mean number of calls per unit time <li><i>If ON-OFF traffic model:</i> Mean OFF period duration of a node in terms of time units </ul>  meanOnDuration <ul> <li><i>If - Poisson traffic model:</i> Expected value for duration of a call in terms of time units <li><i>If ON-OFF traffic model:</i> Expected value for duration of a ON period in terms of time units </ul>  trafficModel Model - for traffic generation  unitTime Scale - of msec during animation |
| Methods | **public void registerNode(**  **Node n)**  Registers a node to the channel.  **Parameters**  n - node that will be registered  **public double generateSNR(**  **Node sensor,**  **int frequency)**  Finds an snr value according to the channel model.  **Parameters**  sensor Node - to assign SNR value  frequency - Frequency to which the sensor senses  **Returns**  snr value  **public double generateSINR(**  **Node transmitter,**  **Node receiver,**  **int freq)**  Finds an snr value according to the channel model.  **Parameters**  transmitter Node - transmitting the signal  receiver Node - to assign SNR value  freq Freqency - which will be used during the communication between transmitter and receiver  **Returns**  snr value at receiver caused by transmitter  **public void occupyFrequency(**  **int frequency,**  **Node n)**  Lets a primary user to occupy a frequency. That is, the primary user starts transmission.  **Parameters**  frequency - In which the user transmits  n Node - that occupies the frequency  **public boolean isOccupied(**  **int freq,**  **int nodeType)**  Re turns whether a frequency is occupied by a given type of node  **Parameters**  freq ID - of frequency  nodeType Type - node, CR or Primary  **Returns**  <ul> <li><b><i>True</i></b> if given frequency is occupied by a given type of node</li> <li><b><i>False</i></b> otherwise</li> </ul>  **public void releaseFrequency(**  **int frequency,**  **Node n)**  Lets a primary user to release a frequency. That is, the primary user finishes transmission.  **Parameters**  frequency In - which the user transmits  n Node - which releases the frequency  **public int freeFrequency()**  Finds a free frequency and returns its ID  **Returns**  ID of frequency. -1 if no free frequency  **public static double dbToMag(**  **double db)**  Computes the magnitude of a given dB  **Parameters**  db - dB value to be computed  **Returns**  magnitude equivalent of db  **public static double magTodb(**  **double mag)**  Computes the dB of a given magnitude  **Parameters**  mag - Magnitude value to be computed  **Returns**  dB equivalent of mag  **public int numberOfFreq()**  Computes the number of frequencies  **Returns**  Number of frequencies  **public double getMeanOnDuration()**  Returns the mean on (active) duration of all (both primary secondary) users.  **Returns**  Mean on duration  **public double getMeanOffDuration()**  Returns the mean off (inactive) duration of all (both primary secondary) users.  **Returns**  Mean off duration  **public int getTrafficModel()**  Returns the traffic model of all (both primary secondary) users.  **Returns**  Traffic Model  **public java.util.ArrayList getFreq(**  **int freq)**  Returns both CR node and primary node using a given frequency  **Parameters**  freq ID - of frequency to find its current users  **Returns**  Arraylist that contains current users of the given frequency |
| Fields | **public static final PRIMARY**  Integer value for Primary node  **public static final CR**  Integer value for CR node  **public static final SIMPLECH**  Simple Channel Model such that SNR = maxSNR / e^f(distance)  **public static final LOGNORMALCH**  Lognormal channel model  **public static final NOFREEFREQ**  There is no available frequency right now  **public maxSNR**  Max SNR value of the channel  **public sinrThreshold**  Minimum SINR threshold to be able to communicate  **public uniform**  Uniform distribution to accomplish frequency assignments  **public static final POISSON**  Poisson traffic model  **public static final ON\_OFF**  On-Off traffic model  **public static unitTime**  Scale of msec during animation |