Cloud Application Development

Week -6

WEEK-6: CLOUDSIM Simulate a cloud scenario using Cloud Sim and run a scheduling algorithm that is not present in Cloud Sim

AIM:

To simulate cloud scenarios using cloudsim and run a scheduling algorithm.

SOFTWARE REQUIRED:

CloudSim 3.0.3, Eclipse IDE

DESCRIPTION:

CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services Originally built primarily at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, the University of Melbourne, Australia, CloudSim has become one of the most popular open source cloud simulators in the research and academia. CloudSim is completely written in Java.

CloudSim is an open-source framework, which is used to simulate cloud computing infrastructure and services. It is developed by the CLOUDS Lab organization and is written entirely in Java. It is used for modelling and simulating a cloud computing environment as a means for evaluating a hypothesis prior to software development in order to reproduce tests and results.

PROCEDURE:

1. Install eclipse and cloud sim 3.0.3 and create a new java project.Give a name of your choice [ClodSimSimmulation].
2. Under the folder go to src right click and create a new package named com.sjfs
3. Right click the package → show in → Explorer. Copy all FCFS code files into this location. (FCFS_SCheduler, FCFsDatacenterBroker etc)
4. Make sure all the files have the same package name you created (ie com.sjfs)
5. Right click the main folder ClodSimSimmulation and go to build path select configure build path.
6. In the new tab go to classpath → add external jar → browse cloudsim jar → apply and close.
7. Right click and run the FCFS_SCheduler to get the output.

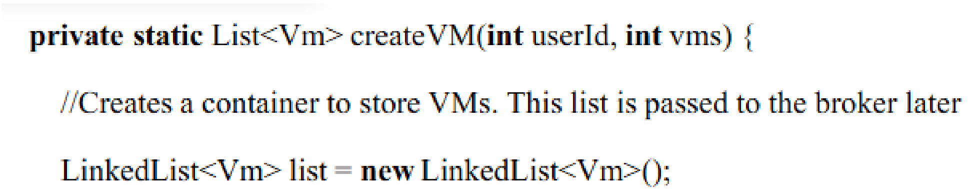
ALGORITHM:

```
1. Input the processes along with their burst time (bt).
2. Find waiting time (wt) for all processes.
3. As the first process that comes need not to wait so
     waiting time for process 1 will be 0 i.e. wt[0] = 0.
4. Find waiting time for all other processes i.e. for
     process i ->
     wt[i] = bt[i-1] + wt[i-1] .
5. Find turnaround time = waiting_time + burst_time
     for all processes.
6. Find average waiting time =
           total_waiting_time / no_of_processes.
7. Similarly, find average turnaround time =
           total_turn_around_time / no_of_processes.
```

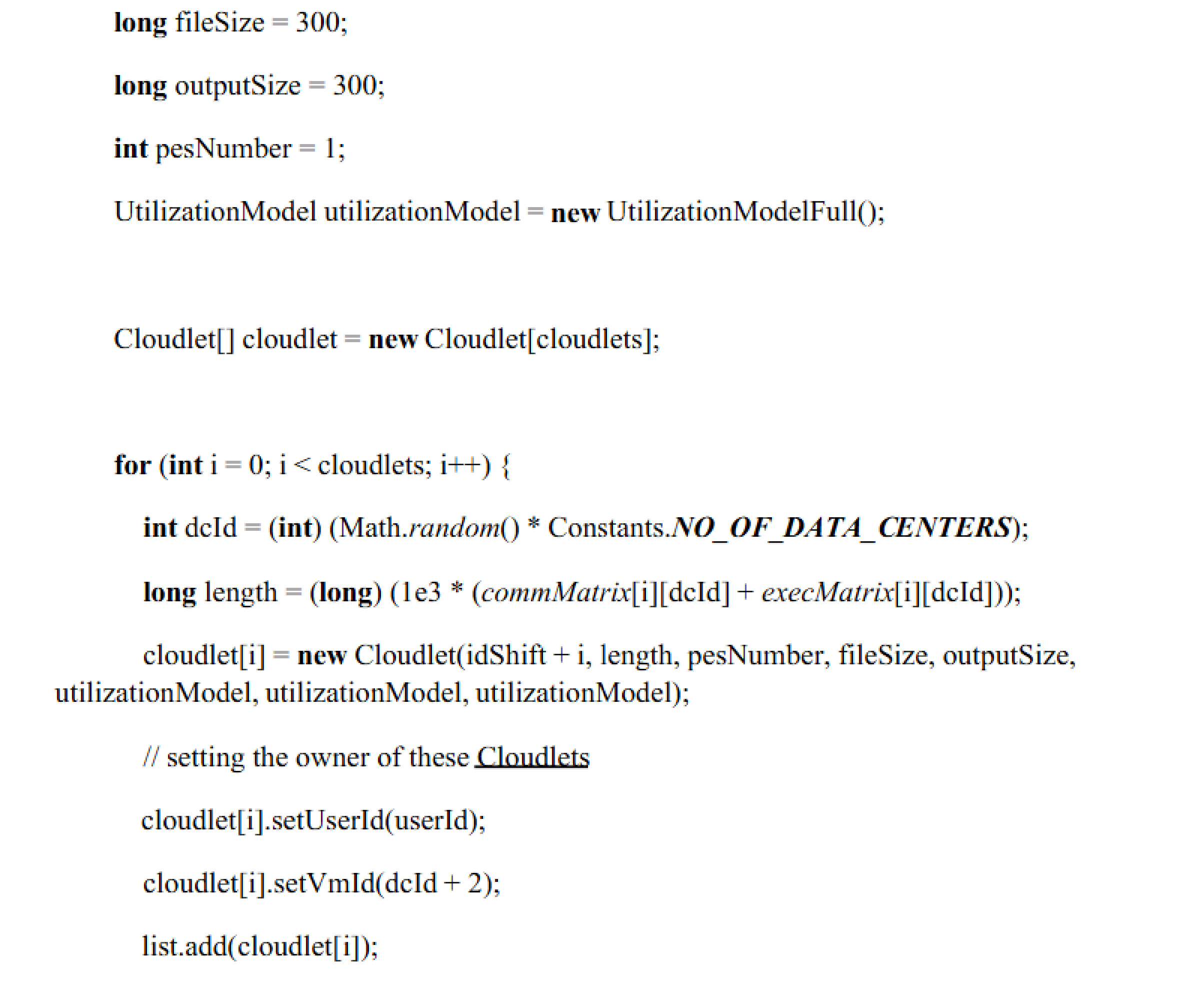
PROGRAM:-

FCFS_SCheduler

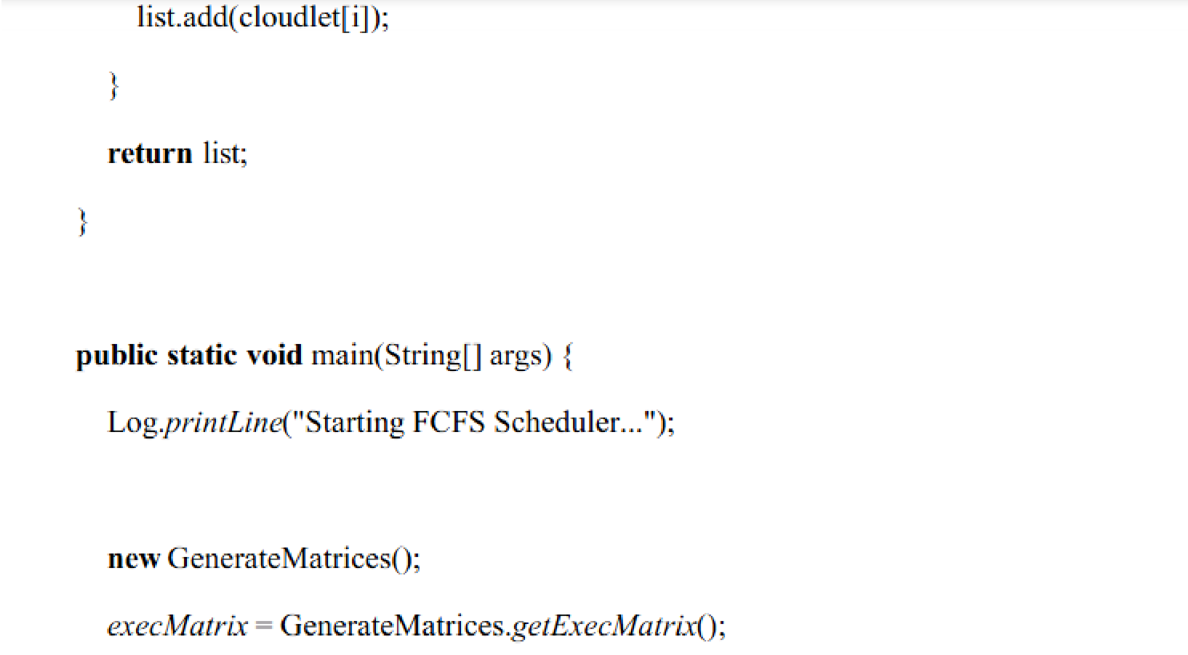
```
 package com.sjfs;

 import org.cloudbus.cloudsim.*;

 import org.cloudbus.cloudsim.core.CloudSim;

 //import utils.Constants;

 //import utils.DatacenterCreator;

 //import utils.GenerateMatrices;



import java.text.DecimalFormat;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;


public class FCFS_Scheduler {


   private static List<Cloudlet> cloudletList;

   private static List<Vm> vmList;

   private static Datacenter[] datacenter;

   private static double[][] commMatrix;

   private static double[][] execMatrix;
```



```
private static List<Vm> createVM(int userId, int vms) {

    //Creates a container to store VMs. This list is passed to the broker later

    LinkedList<Vm> list = new LinkedList<Vm>();
//VM Parameters
long size = 10000; //image size (MB)
int ram = 512; //vm memory (MB)
int mips = 250;
long bw = 1000;
int pesNumber = 1; //number of cpus
String vmm = "Xen"; //VMM name

//create VMs
Vm[] vm = new Vm[vms];

for (int i = 0; i < vms; i++) {
    vm[i] = new Vm(datacenter[i].getId(), userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());
    list.add(vm[i]);
}
    }

    return list;
}

private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift) {
    // Creates a container to store Cloudlets
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

    //cloudlet parameters
```



```
        long fileSize = 300;

        long outputSize = 300;

        int pesNumber = 1;

        UtilizationModel utilizationModel = new UtilizationModelFull();


        Cloudlet[] cloudlet = new Cloudlet[cloudlets];


        for (int i = 0; i < cloudlets; i++) {

            int dcId = (int) (Math.random() * Constants.NO_OF_DATA_CENTERS);

            long length = (long) (1e3 * (commMatrix[i][dcId] + execMatrix[i][dcId]));

            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize,
    utilizationModel, utilizationModel, utilizationModel);

            // setting the owner of these Cloudlets

            cloudlet[i].setUserId(userId);

            cloudlet[i].setVmId(dcId + 2);

            list.add(cloudlet[i]);
```



```
            list.add(cloudlet[i]);

        }

        return list;

    }


    public static void main(String[] args) {

        Log.printLine("Starting FCFS Scheduler...");


        new GenerateMatrices();

        execMatrix = GenerateMatrices.getExecMatrix();
```

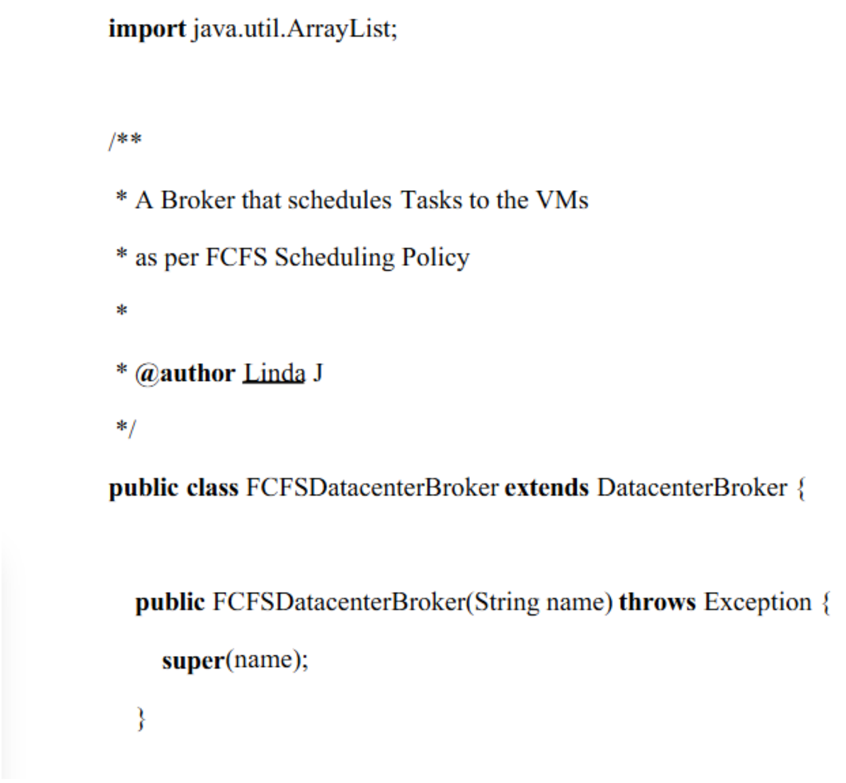
```
commMatrix = GenerateMatrices.getCommMatrix();

try {
    int num_user = 1;   // number of grid users
    Calendar calendar = Calendar.getInstance();
    boolean trace_flag = false;  // mean trace events

    CloudSim.init(num_user, calendar, trace_flag);

    // Second step: Create Datacenters
    datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
    for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
        datacenter[i] = DatacenterCreator.createDatacenter("Datacenter_" + i);
    }

      //Third step: Create Broker
      FCFSDatacenterBroker broker = createBroker("Broker_0");
      int brokerId = broker.getId();

      //Fourth step: Create VMs and Cloudlets and send them to broker
      vmList = createVM(brokerId, Constants.NO_OF_DATA_CENTERS);
      cloudletList = createCloudlet(brokerId, Constants.NO_OF_TASKS, 0);

      broker.submitVmList(vmList);
      broker.submitCloudletList(cloudletList);
    // Fifth step: Starts the simulation
    CloudSim.startSimulation();

    // Final step: Print results when simulation is over
    List<Cloudlet> newList = broker.getCloudletReceivedList();
    //newList.addAll(globalBroker.getBroker().getCloudletReceivedList());

    CloudSim.stopSimulation();

    printCloudletList(newList);

    Log.printLine(FCFS_Scheduler.class.getName() + " finished!");
} catch (Exception e) {
    e.printStackTrace();
    Log.printLine("The simulation has been terminated due to an unexpected error");
}
            }
        }

        private static FCFSDatacenterBroker createBroker(String name) throws Exception {
            return new FCFSDatacenterBroker(name);
        }

        /**
         * Prints the Cloudlet objects
```

```
     */
    private static void printCloudletList(List<Cloudlet> list) {
      int size = list.size();
      Cloudlet cloudlet;

      String indent = "    ";
      Log.printLine();
      Log.printLine("========== OUTPUT ==========");
      Log.printLine("Cloudlet ID" + indent + "STATUS" +
          indent + "Data center ID" +
          indent + "VM ID" +
          indent + indent + "Time" +
          indent + "Start Time" +
          indent + "Finish Time");

DecimalFormat dft = new DecimalFormat("###.##");
dft.setMinimumIntegerDigits(2);
for (int i = 0; i < size; i++) {
  cloudlet = list.get(i);
  Log.print(indent + dft.format(cloudlet.getCloudletId()) + indent + indent);

  if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
    Log.print("SUCCESS");
```

```
      Log.printLine(indent + indent + dft.format(cloudlet.getResourceId()) +
          indent + indent + indent + dft.format(cloudlet.getVmId()) +
          indent + indent + dft.format(cloudlet.getActualCPUTime()) +
          indent + indent + dft.format(cloudlet.getExecStartTime()) +
          indent + indent + indent + dft.format(cloudlet.getFinishTime()));
    }
  }
  double makespan = calcMakespan(list);
  Log.printLine("Makespan using FCFS: " + makespan);
}

private static double calcMakespan(List<Cloudlet> list) {
  double makespan = 0;
  double[] dcWorkingTime = new double[Constants.NO_OF_DATA_CENTERS];

      for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
        int dcId = list.get(i).getVmId() % Constants.NO_OF_DATA_CENTERS;
        if (dcWorkingTime[dcId] != 0) --dcWorkingTime[dcId];
        dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];
        makespan = Math.max(makespan, dcWorkingTime[dcId]);
      }
      return makespan;
    }
  }
```

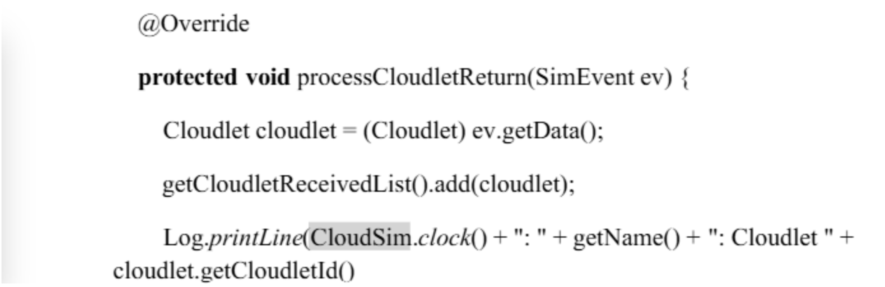
FCFsDatacenterBroker

```
package com.sjfs;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.core.SimEvent;
```



```
import java.util.ArrayList;

/**
 * A Broker that schedules Tasks to the VMs
 * as per FCFS Scheduling Policy
 *
 * @author Linda J
 */
public class FCFSDatacenterBroker extends DatacenterBroker {

  public FCFSDatacenterBroker(String name) throws Exception {
    super(name);
  }
```

```
   //scheduling function
   public void scheduleTaskstoVms() {
        ArrayList<Cloudlet> clist = new ArrayList<Cloudlet>();


        for (Cloudlet cloudlet : getCloudletSubmittedList()) {
            clist.add(cloudlet);
        }


        setCloudletReceivedList(clist);
    }

```

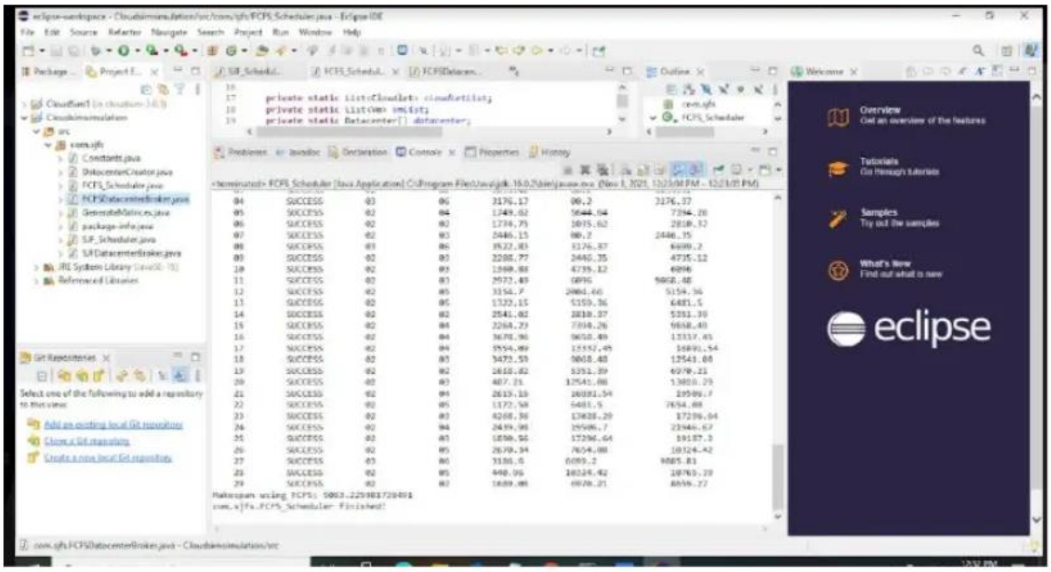


```
    @Override
    protected void processCloudletReturn(SimEvent ev) {
        Cloudlet cloudlet = (Cloudlet) ev.getData();
        getCloudletReceivedList().add(cloudlet);
        Log.printLine(CloudSim.clock() + ": " + getName() + ": Cloudlet " +
cloudlet.getCloudletId()
```

```
cloudlet.getCloudletId()
            + " received");
        cloudletsSubmitted--;
        if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) {
            scheduleTaskstoVms();
            cloudletExecution(cloudlet);
        }
    }
```

```
protected void cloudletExecution(Cloudlet cloudlet) {

  if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) { // all cloudlets executed
    Log.printLine(CloudSim.clock() + ": " + getName() + ": All Cloudlets executed. Finishing...");
    clearDatacenters();
    finishExecution();
  } else { // some cloudlets haven't finished yet
    if (getCloudletList().size() > 0 && cloudletsSubmitted == 0) {
      // all the cloudlets sent finished. It means that some bount
      // cloudlet is waiting its VM be created
      clearDatacenters();
      createVmsInDatacenter(0);
    } } }}
```

OUTPUT:



```
private static List<Cloudlet> cloudletList;
private static List<Vm> vmlist;
private static Datacenter[] datacenter;
```

```
<terminated> FCFS_Scheduler [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (Nov 1, 2021, 12:23:04 PM – 12:23:05 PM)
    04        SUCCESS       03          06        3176.17      00.2         3176.37
    05        SUCCESS       02          04        1749.62      5644.64      7394.26
    06        SUCCESS       02          02        1774.75      1035.62      2810.37
    07        SUCCESS       02          03        2446.15      00.2         2446.35
    08        SUCCESS       03          06        3522.83      3176.37      6699.2
    09        SUCCESS       02          03        2288.77      2446.35      4735.12
    10        SUCCESS       02          03        1360.88      4735.12      6096
    11        SUCCESS       02          03        2572.48      6096         8668.48
    12        SUCCESS       02          05        3154.7       2004.66      5159.36
    13        SUCCESS       02          05        1322.15      5159.36      6481.5
    14        SUCCESS       02          02        2541.02      2810.37      5351.39
    15        SUCCESS       02          04        2264.23      7394.26      9658.49
    16        SUCCESS       02          04        3678.96      9658.49      13337.45
    17        SUCCESS       02          04        3554.09      13337.45     16891.54
    18        SUCCESS       02          03        3472.55      9068.48      12541.03
    19        SUCCESS       02          02        1618.82      5351.39      6970.21
    20        SUCCESS       02          03        487.21       12541.03     13028.23
    21        SUCCESS       02          04        2615.16      16891.54     19506.7
    22        SUCCESS       02          05        1172.58      6481.5       7654.08
    23        SUCCESS       02          03        4268.36      13028.23     17296.64
    24        SUCCESS       02          04        2439.96      19506.7      21946.67
    25        SUCCESS       02          03        1850.56      17296.64     19147.2
    26        SUCCESS       02          05        2670.34      7654.08      10324.42
    27        SUCCESS       03          06        3186.6       6699.2       9885.81
    28        SUCCESS       02          05        440.96       10324.42     10765.38
    29        SUCCESS       02          02        1689.08      6970.21      8659.29
Makespan using FCFS: 5063.225981720491
com.sjfs.FCFS_Scheduler finished!
```

eclipse-workspace - CloudsimSimulation/src/com/sjfs/FCFS_Scheduler.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package ... Project E...
SJF_Schedul... FCFS_Schedul... FCFSDatacen...
Outline
com.sjfs
FCFS_Scheduler
Welcome
CloudSim1
CloudsimSimulation
src
com.sjfs
Constants.java
DatacenterCreator.java
FCFS_Scheduler.java
FCFSDatacenterBroker.java
GenerateMatrices.java
package-info.java
SJF_Scheduler.java
SJFDatacenterBroker.java
JRE System Library
Referenced Libraries
Git Repositories
Select one of the following to add a repository to this view:
Add an existing local Git repository
Clone a Git repository
Create a new local Git repository
Problems Javadoc Declaration Console Properties History
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
eclipse
com.sjfs.FCFSDatacenterBroker.java - CloudsimSimulation/src

RESULT:

Simulation of cloud scenarios using cloudsim was done successfully and a scheduling algorithm was compiled.